GÉNIE LOGICIEL FORMEL:
THÉORIE, PRATIQUE ET PÉDAGOGIE
(APPRENTISSAGE À PARTIR DE L'EXPÉRIENCE DU VOTE ÉLECTRONIQUE)

Mémoire présenté en vue de l'obtention de
L'Habilitation à Diriger des Recherches

Université Henri Poincaré, Nancy I, France

Présentée par

J. Paul Gibson (BSc, PhD, Maître de Conférences)
Le département Logiciels-Réseaux (LOR),
Telecom & Management SudParis,
9 rue Charles Fourier, 91011 Évry cedex.

COMITÉ

Rapporteurs :

1. Prof. Yamine Ait-Ameur, LISI / ENSMA.
2. Prof. Pascale Le Gall, Université d'Evry-Val d'Essonne.
3. Prof. Tiziana Margaria, Universität Potsdam.

Examinateurs :

1. Prof. Ana Cavalli, TSP Evry.
2. Prof. Michael Butler, University of Southampton.
3. Prof. Dominique Méry, UHP Nancy.

No part of a book is so intimate as the Preface. Here, . . . , the author descends from his platform, and speaks with his reader as man to man, disclosing his hopes and fears, seeking sympathy for his difficulties, offering defence or defiance, according to his temper, against the criticisms which he anticipates. It thus happens that a personality which has been veiled by a formal method throughout many chapters, is suddenly seen face to face in the Preface;

Charles W. Eliot, [Prefaces and Prologues to Famous Books with Introductions, Notes and Illustrations]

PREFACE

I have recently been involved in the domain of electronic voting (e-voting). I had the pleasure, and responsibility, of being an expert advisor to the Commission for Electronic Voting in Ireland[1] Through my work on this commission, and subsequent research into the e-voting problem, I realised that software in these systems (and in many other similar systems) was very poorly engineered. Further, errors were made in the development of these systems that could have been avoided if an experienced, professional, well-educated software engineer had been involved. Finally, I saw that e-voting systems provided a complex challenge to the formal methods community: could we demonstrate where and how the application of formal techniques would have made a difference?

My personal approach to research is founded on the belief that theoretical and applied research are symbiotic; and that researchers, where possible, should avoid separating the two. Further, I firmly believe that research and teaching are also symbiotic — research should inform and be informed by the teaching process.

E-voting provides an excellent example of this symbiotic triangle: understanding the problem requires understanding fundamental boundaries and challenges for software engineers which can only be addressed through advances in software engineering theory; the practical implementation of e-voting machines has failed because existing, well-established, software engineering methods have not been applied; and e-voting provides an excellent case study for educating the next generation of software engineers.

In this HDR my objective is to demonstrate my different contributions as a researcher in the domain of formal software engineering: in fundamental theory, applied research and in education. It is 20 years since my research career started (as a PhD student) and this HDR provides an excellent opportunity for me to look back in order to look forward.

$\overline{JPaulGibson}$

A note on language: this HDR was originally written in English. Rather than completely translating the text into a second version in French, key sections — the ABSTRACT, INTRODUCTION and CONCLUSIONS — have been completely translated into French and incorporated directly (in parallel with the English source text) into the HDR. Translated summaries of key subsections in sections 3 and 4 — the RESEARCH CONTRIBUTION and RESEARCH PROPOSAL — have also been included.

[1] The Independent Commission on Electronic Voting and Counting at Elections was established by the Government of Ireland, March, 2004 and dissolved in September, 2006 (see `www.cev.ie`).

There is no abstract art. You must always start with something. Afterward you can remove all traces of reality.
Pablo Picasso

Computer Science is a science of abstraction — creating the right model for a problem and devising the appropriate mechanizable techniques to solve it.
A. Aho and J. Ullman

ABSTRACT

This HDR is a summary of the last 20 years of my research career (since I started my PhD). It provides an overview of all my research activities. It focuses on my contributions to the domain of formal software engineering. With respect to fundamental theory, it reports on my work in the area of feature interactions and feature composition. To demonstrate my ability to apply research to real-world problems, it reports on my research on electronic voting. It then reports on my research into the symbiotic nature of research and teaching, with emphasis on formal methods and problem based learning. Finally, it puts forward a proposal for research into a software product line for e-voting systems, that combines theory, practice and education in a complementary manner.

RÉSUMÉ

L'HDR est un condensé de nos 20 dernières années de carrière dans la recherche (depuis que nous avons commencé notre PhD). Elle fournit une vue d'ensemble de toutes nos activités de recherche. Elle se focalise sur nos contributions dans le domaine du génie logiciel formel. Au niveau de la théorie fondamentale, elle rend compte de notre travail dans le domaine des interactions de service et de leur composition. Afin de démontrer notre capacité à appliquer la recherche aux problèmes du monde réel, elle expose notre recherche sur le vote électronique. Elle fait également un compte rendu de notre recherche sur la nature symbiotique de la recherche et de l'enseignement, avec une accentuation sur les méthodes formelles et l'apprentissage par problèmes. Pour finir, elle met en avant une proposition de recherche sur une ligne de produit logiciel pour les systèmes de vote électronique qui combinent de façon complémentaire théorie, pratique et pedagogie.

I can no other answer make but thanks,
And thanks, and ever thanks,
William Shakespeare, [Twelfth Night, Act 3]

Celui qui dans la vie est parti de zéro pour n'arriver à rien
dans l'existence n'a de merci à dire à personne.
Pierre Dac , [Les Pensées]

THANKS

Sincere thanks to my wife, Patricia; and my two girls, Charlotte and Juliette.
Without you I would never have started (nor finished) this HDR!

Contents

1 INTRODUCTION: — THEMATIQUE DE RECHERCHE

It should be noted that no ethically-trained software engineer would ever consent to write a "DestroyBaghdad" procedure. Basic professional ethics would instead require him to write a "DestroyCity" procedure, to which "Baghdad" could be given as a parameter.
Nathaniel S. Borenstein

Il avait acheté ma voix. J'ai trouvé ça si malhonnête que j'ai voté pour l'autre.
Yvan Audouard

1.1 Preliminary Questions — Questions préliminaires

1.1.1 What Is Software Engineering? — Qu'est-ce que le génie logiciel ?

All engineering requires discipline in order to construct solutions to real-world problems; and the foundation of this discipline must be scientific rigour. Science is the synthesis and analysis of mathematical models, based on observation and experiment, used to capture properties, and define abstractions, of the observed world. Thus, modelling and abstraction are fundamental to engineering.

Software engineering can be regarded as the establishment and application of sound principles and methods in the development of software for execution on computers[PGL00]. Computer Science is the study of the mechanical processes that permit the representation, processing, storage and communication of information. Thus, software engineers must work with models and abstractions of computers, computations and information[Har92].

Toute ingénierie requiert de la discipline afin de construire des solutions aux problèmes du monde réel ; et le fondement de cette discipline doit être la rigueur scientifique. La science, c'est la synthèse et l'analyse de modèles mathématiques basées sur l'observation et sur l'expérience et utilisées pour capturer les propriétés et pour définir les abstractions du monde observé. Ainsi, la modélisation et l'abstraction sont fondamentales pour l'ingénierie.

Le génie logiciel peut être envisagé comme l'établissement et l'application de principes solides et de méthodes, dans le développement de logiciels pour l'exécution sur ordinateurs [PGL00]. L'informatique, c'est l'étude de procédés mécaniques qui permettent la représentation, le traitement, le stockage et la communication de l'information. Ainsi, les ingénieurs de logiciels doivent travailler avec modèles et abstractions d'ordinateurs, calculs et information [Har92].

1.1.2 Does Software Engineering Lack Discipline? — Le génie logiciel manque-t-il de discipline ?

Software engineering lacks discipline [CGM+89] for two complementary reasons. Firstly, there are only a few well-established principles that bridge the gap between computer science and software engineering. Secondly, where such principles ex-

Le génie logiciel manque de discipline pour deux raisons complémentaires [CGM+89]. Premièrement, il n'y a que peu de principes bien établis faisant le lien entre informatique et génie logiciel. Deuxièmement, là où de tels principes existent, les ingénieurs pratiquant

ist practising engineers often fail to apply them.

Formal methods incorporate the principles of moving from abstract models of problems to be solved (the *what*) to concrete models of solutions involving computers (the *how*). They are founded on mathematical models of computability, computational complexity, communication and correctness. Yet, few practising software engineers know when or how to apply these methods. It is therefore not surprising that the quality of software systems is compromised and that such systems often fail to correctly solve the problem for which they were developed.

1.1.3 What Is Electronic Voting? — Qu'est-ce que le vote électronique ?

Electronic voting (e-voting), in general, refers to any electronic means of casting a vote and/or electronic means of counting votes. It can also involve electronic transmission of information necessary for the process of vote casting and/or counting during an election.

Applying state-of-the-art computer and information technology to "modernise" the voting process has the potential to make improvements over the existing paper (or mechanical) systems; but it also introduces new concerns with respect to secrecy, accuracy and security [Gri03]. The debate over the advantages and disadvantages of e-voting is not a new one; and recent use of such systems in actual elections has led to their analysis from a number of viewpoints: usability [HBL+05], trustworthiness and safety criticality [MG03], and risks and threats [Neu90].

The potential advantages are generally accepted, for example: faster result tabulation, elimination of human error which occurs in manual vote tabulation, assistance to voters with "special" needs, defence against fraudulent practices (e.g. with postal votes[Bro05]), and improving the "fairness" of count systems that incorporate "unfair" non-deterministic procedures[TR00].

manquent souvent de les appliquer.

Les méthodes formelles incorporent les principes du passage de modèles abstraits de problèmes à résoudre (le quoi) vers des modèles concrets de solutions impliquant des ordinateurs (le comment). Elles sont fondées sur des modèles mathématiques de computability, complexité, de communication et de justesse. Cependant, peu d'ingénieurs en logiciels qui pratiquent savent quand ou comment appliquer ces méthodes. Il n'est donc pas surprenant que la qualité des systèmes de logiciels soit compromise et que de tels systèmes échouent souvent à résoudre correctement le problème pour lequel ils ont été développés.

Le vote électronique se réfère en général à tout moyen électronique de voter et/ou à tout moyen pour compter les votes. Cela peut également impliquer la transmission électronique des informations nécessaires au traitement du vote et/ou du dépouillement du scrutin pendant une élection.

Utiliser un ordinateur de pointe avec la technologie d'information pour "moderniser" le processus de vote électronique a le potentiel d'apporter du progrès sur les systèmes existants avec papier (ou mécaniques) ; mais ils génèrent aussi de nouvelles inquiétudes par rapport au secret, à la précision et à la sécurité [Gri03]. Le débat autour des avantages et des inconvénients du vote électronique n'est pas nouveau ; et l'utilisation de tels systèmes lors de réelles élections a conduit à leur analyse selon divers points de vue : usability [HBL+05], fiabilité et "safety criticality" [MG03], et risques et menaces [Neu90].

Les avantages potentiels sont généralement acceptés, comme par exemple : tabulation plus rapide des résultats, élimination de l'erreur humaine se produisant lors d'une tabulation par vote manuel, assistance aux votants ayant un handicap, défense contre des pratiques frauduleuses (comme par exemple pour le vote par correspondance [Bro05]), et amélioration de la "justesse" des systèmes de comptage qui incorporent des procé-

Despite ever-increasing uncertainty over the trustworthiness of these systems — which is one of the major disadvantages associated with them — many countries (particularly in Europe[SL03]) have recently chosen to adopt e-voting. The main risks that have been clearly identified seem not to concern those responsible for procuring the systems. In fact, it appears that e-voting is just one, well-publicised, example of governments wishing to adopt new technologies[SE03] before the risks and benefits, as perceived by the public[HKG07], have been properly analysed and debated.

dures "injustes" non-déterministes [TR00].

En dépit d'une incertitude incessante sur la fiabilité de ces systèmes — ceci étant leur inconvénient majeur — de nombreux pays ont choisi d'adopter le vote électronique [SL03]. Les principaux risques clairement identifiés paraissent ne pas inquiéter ceux-là mêmes responsables de procurer ces systèmes. En fait, le vote électronique semble simplement être un exemple très médiatisé du souhait de certains gouvernements à vouloir adopter de nouvelles technologies [SE03] sans qu'au préalable les risques et les bénéfices (ainsi perçus par le public [HKG07]) n'aient été analysés et débattus.

1.1.4 What Can We Learn From E-voting?
— Que pouvons-nous apprendre du vote électronique ?

Electronic voting systems provide excellent examples of poorly engineered software. Analysis of such systems can, and should, help us to identify where engineers failed to apply fundamental (software) engineering principles. In particular, they provide an opportunity to illustrate where the application of formal methods could have helped the developers to have constructed much better quality solutions to the e-voting problem[CGM07a, CGM07b].

In this HDR we analyse e-voting and show that many problems that arose could have been avoided if rigorous software engineering practices, including formal methods, had been applied. We argue that educators, and the education system, are as responsible for poorly built software systems as the (so called) engineers who developed them. Finally, we argue that the next generation of electronic voting systems need to be better engineered than the current generation; and we propose that the most promising approach to achieving this requires the development of a *formal software product line*.

Les systèmes de vote électronique fournissent d'excellents exemples de logiciels pauvrement construits. L'analyse de tels systèmes peut, et devrait, aider à identifier là où les ingénieurs ont échoué à appliquer les principes fondamentaux du génie logiciel. Plus spécialement, ils fournissent une bonne illustration de là où l'application des méthodes formelles aurait pu aider les développeurs à construire des solutions de bien meilleure qualité au problème du vote électronique [CGM07a, CGM07b].

Dans cette HDR, nous analysons le vote électronique et montrons que de nombreux problèmes rencontrés auraient pu être évités si les pratiques rigoureuses du génie logiciel incluant les méthodes formelles avaient été appliquées. Nous argumentons que les enseignants, ainsi que le système éducatif, sont tout autant responsables de la pauvre construction de systèmes logiciels que les (soi-disant) ingénieurs les ayant développés. Au final, nous argumentons que la prochaine génération des systèmes de vote électronique nécessite une meilleure construction que la génération actuelle ; et nous suggérons que l'approche la plus prometteuse pour atteindre cela requiert le développement d'une *ligne de produit de logiciels formels*.

1.1.5 What Can We Learn (About Teaching Software Engineering) From E-voting?
— Que pouvons-nous apprendre (sur l'enseignement du génie logiciel) du vote électronique ?

E-voting has a wide range of different types

Le vote électronique possède un large éventail de

of requirements [Gib00] that make it ideal for teaching about complex software systems [AM05, BF07]. E-voting provides an ideal opportunity to present ethical issues. All students are familiar with voting and problems with e-voting systems, and the subject is one which both interests and motivates them. It provides an excellent study for problem based learning throughout a software engineering programme[Gib08b].

différents types d'exigences [Gib00] qui le rendent idéal pour l'enseignement des systèmes complexes de logiciels[AM05, BF07]. Le vote électronique offre une opportunité idéale d'aborder des questions d'ordre éthique. Tous les étudiants connaissent bien le vote et les problèmes liés aux systèmes de vote électronique, et c'est un sujet qui les intéresse et les motive à la fois. Cela apporte une excellente étude pour l'apprentissage par problèmes à travers un programme de génie logiciel [Gib08b].

1.1.6 Why A Software Product Line? — Pourquoi une ligne de produit de logiciels?

The software in e-voting machines has not, in general, been well-engineered[GM08]. Many governments have chosen to adopt e-voting as a show-case for innovative technology[SL03]. It is a poor reflection on the profession of software engineering that the software in these systems is, in general, neither trusted nor trustworthy[RR06]. We propose that the software engineering community should look upon this as an opportunity to demonstrate just how much software engineering methods, techniques and tools have evolved since the turn of the century[PGL00]; and that the software industry is now mature enough to develop e-voting machines that are highly dependent on software and that are highly dependable.

Software Product Lines (SPLs) [CN02] are attracting attention in the area of applied software engineering research. The challenge is to demonstrate how and why an e-voting SPL could be built. E-voting systems correspond in terms of size and complexity to those reported in a number of SPL case studies [Bos99b]. The number of variations across systems[SP06] is large enough to merit an SPL approach, but not so large as to be unmanageable. Furthermore, these systems exhibit a large amount of common functionality and so the potential for re-use is high. The aspect of e-voting that may be more challenging is that the software may be considered (safety or mission) critical [MG03].

La majorité du temps, le logiciel des machines de vote électronique n'a pas été correctement construit [GM08]. De nombreux gouvernements ont choisi d'adopter le vote électronique comme modèle d'innovation technologique [SL03]. Le fait que le logiciel au sein de ces systèmes soit la plupart du temps ni fiable, ni de confiance [RR06] reflète de façon négative sur la profession d'ingénieur en génie logiciel.

Nous estimons que l'ensemble de la profession du génie logiciel doit saisir cette opportunité pour démontrer simplement comment les méthodes, techniques et outils fournis par le génie logiciel ont évolué depuis le début du siècle [PGL00], et que l'industrie du logiciel est dorénavant assez avancée pour développer des machines de vote électronique qui soient à la fois hautement dépendantes des logiciels et hautement dignes de confiance.

Les *Software Product Lines* (SPLs) [CN02] attirent actuellement l'attention dans le domaine de la recherche en génie logiciel appliqué. Le défi est de prouver comment et pourquoi une *e-voting SPL* pourrait être construite. Les systèmes de vote électronique correspondent en termes de taille et de complexité à ceux rapportés dans un certain nombre de cas d'études SPL [Bos99b]. Le nombre de variations à travers les systèmes [SP06] est assez large pour mériter une approche SPL, mais pas trop large pour être ingérable. En outre, ces systèmes manifestent une large quantité de fonctionnalités communes, donnant ainsi un potentiel de réutilisation élevé. L'aspect du vote électronique pouvant davantage représenter un challenge

However, recent research suggests that SPLs can be used to develop safety critical systems [Liu07].

Many of the problems that have arisen in the domain of e-voting have arisen because of poorly specified requirements and standards documents [MG06]. It has been proposed that a comprehensive domain analysis be carried out before standards are re-engineered [GM08]. The resulting domain models should provide the foundations upon which standards could be built; and they would also play a major role in the development of an e-voting SPL.

The development of a formal SPL for evoting is a very challenging research area. In particular, the feature interaction problem — particularly during requirements modelling[Gib97] — will require the integration of different modelling languages in some sort of unified semantic framework[GMM97].

1.2 Structure of HDR — Structure de l'HDR

In section 2, I provide an overview of all my research activities in the last 20 years (and not just those activities directly relevant to the main body of this work). This section should set the context for the specific research contributions that I present in detail in section 3; and should demonstrate my competencies with respect to the research proposed in section 4

In section 3, I review my research contributions in the three complementary areas:

1. Software Engineering Fundamental Theory: Feature Composition

2. Software Engineering Applied Theory: E-voting Systems

3. Software Engineering Education: Formal Methods and Problem-Based Learning

Section 4 proposes that the current best practice (and state-of-the-art) in software engineering ,appropriate to the e-voting problem, would be to

est le fait que le logiciel peut être considéré "criticial" [MG03]. Cependant, de récentes recherches suggèrent que les SPLs peuvent être utilisées pour développer des systèmes critiques de sécurité [Liu07].

Nombre de problèmes survenus dans le domaine du vote électronique l'ont été en raison de besoins mal spécifiés [MG06]. On a proposé qu'une analyse complète et détaillée du domaine soit entreprise avant que les standards ne soient reconstruits [GM08]. Les modèles du domaine en résultant devraient fournir des fondations sur lesquelles les standards pourraient être construits ; et ils joueraient également un rôle majeur dans le développement d'une SPL de vote électronique.

Le développement d'une SPL formelle pour vote électronique est un domaine de recherche comportant un grand challenge ; en particulier, le *feature interaction problem* — surtout pendant la modélisation des besoins [Gib97] — qui exigera l'intégration de différents langages de modélisation dans une sorte de cadre sémantique unifié [GMM97].

Dans la section 2, nous fournissons une vue d'ensemble de toutes nos activités de recherche depuis ces 20 dernières années (et pas seulement les activités liées au sujet principal de ce travail). Cette section établit le contexte pour les contributions spécifiques de recherche que nous présentons de façon détaillée en section 3 et montre nos compétences par rapport à la recherche proposée, en section 4.

En section 3, nous passons en revue nos contributions à la recherche dans trois domaines complémentaires :

1. Théorie Fondamentale du Génie Logiciel : Composition de services

2. Théorie Appliquée du Génie Logiciel : Systèmes de Vote Electronique

3. Enseignement du Génie Logiciel : Méthodes Formelles et l'apprentissage par problèmes

La section 4 avance qu'actuellement la meilleure pratique (de pointe) en génie logiciel, appropriée au

combine formal methods with software product lines (SPLs) in order to be able to verify the correct behaviour of a family of voting systems with common sets of requirements but each having their own unique combination of features.

In section 5 we conclude the thesis by making predictions about the future of software development, in general, and possible future research of the author, in particular.

problème du vote électronique, serait de combiner les méthodes formelles avec les "software product lines" (SPLs), afin de pouvoir vérifier l'exactitude du comportement d'une famille de systèmes de vote, avec des ensembles communs de besoins — chacun ayant cependant leur propre et unique combinaison de serives ("features").

En section 5, nous concluons l'HDR en établissant des prédictions sur le futur du développement logiciel en général, et sur les possibilités de recherche de l'auteur, plus particulièrement.

1.A-Introduction: Bibliography

[AM05] Chris Armen and Ralph Morelli. Teaching about the risks of electronic voting technology. *SIGCSE Bull.*, 37(3):227–231, 2005.

[BF07] Matt Bishop and Deborah A. Frincke. Achieving learning objectives through e-voting case studies. *IEEE Security and Privacy*, 5(1):53–56, 2007.

[Bos99] Jan Bosch. Product-line architectures in industry: a case study. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 544–554, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[Bro05] Ian Brown. Who is enfranchised by remote voting? In *COMPSAC '05: Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05) Volume 1*, page 500, Washington, DC, USA, 2005. IEEE Computer Society.

[CGM+89] D. E. Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R. Young. Computing as a discipline. *Commun. ACM*, 32(1):9–23, 1989.

[CGM07a] Dominique Cansell, J. Paul Gibson, and Dominique Méry. Formal verification of tamper-evident storage for e-voting. In *SEFM*, pages 329–338. IEEE Computer Society, 2007.

[CGM07b] Dominique Cansell, J. Paul Gibson, and Dominique Méry. Refinement: A constructive approach to formal software design for a secure e-voting interface. *Electr. Notes Theor. Comput. Sci.*, 183:39–55, 2007.

[CN02] P. Clements and L. Northrop. *Software product lines*. Addison-Wesley Boston, 2002.

[Gib97] J. Paul Gibson. Feature requirements models: Understanding interactions. In Petre Dini, Raouf Boutaba, and Luigi Logrippo, editors, *Feature Interactions in Telecommunications Networks IV, (FIW 1997)*, pages 46–60, Montréal, Canada, 1997. IOS Press.

[Gib00] J. Paul Gibson. Formal requirements engineering: Learning from the students. In Doug Grant, editor, *12th Australian Software Engineering Conference (ASWEC 2000)*, pages 171–180. IEEE Computer Society, 2000.

[Gib08] J. Paul Gibson. Weaving a formal methods education with problem-based learning. In T. Margaria and B. Steffen, editors, *3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*, volume 17 of *Communications in Computer and Information Science (CCIS)*, pages 460–472, Porto Sani, Greece, October 2008. Springer-Verlag, Berlin Heidelberg.

[GM08] J. Paul Gibson and Margaret McGaley. Verification and maintenance of e-voting systems and standards. In Dan Remenyi, editor, *8th European Conference on e-Government*, pages 283–289. Academic Publishing International, July 2008. ISBN 978-1-906638-09-2.

[GMM97] J. Paul Gibson, Bruno Mermet, and Dominique Méry. Feature interactions: A mixed semantic model approach. In Henry McGloughlin and Gerard O'Regan, editors, *1st Irish Workshop on Formal Methods (IWFM 1997)*, Electronic Workshops in Computing, Dublin, Ireland, July 1997. BCS.

[Gri03] Dimitris Gritzalis, editor. *Secure Electronic Voting*, volume 7 of *Advances in Information Security*. Kluwer Academic, 2003.

[Har92] David Harel. Biting the silver bullet - toward a brighter future for system development. *IEEE Computer*, 25(1):8–20, 1992.

[HBL+05] Paul S. Herrnson, Benjamin B. Bederson, Bongshin Lee, Peter L. Francia, Robert M. Sherman, Frederick G. Conrad, Michael Traugott, and Richard G. Niemi. Early appraisals of electronic voting. *Social Science Computer Review*, 23(3):274–292, 2005.

[HKG07] Mark Horst, Margôt Kuttschreuter, and Jan M. Gutteling. Perceived usefulness, personal experiences, risk perception and trust as determinants of adoption of e-government services in The Netherlands. *Computers in human behavior*, 23(4):1838–1852, 2007.

[Liu07] Jing Liu. Handling safety-related feature interaction in safety-critical product lines. In *ICSE Companion*, pages 85–86. IEEE Computer Society, 2007.

[MG03] Margaret McGaley and J. Paul Gibson. E-voting: a safety critical system. Report NUIM-CS-TR-2003-2, Department of Computer Science, National University of Ireland, Maynooth, 2003.

[MG06] Margaret McGaley and J. Paul Gibson. A critical analysis of the council of europe recommendations on e-voting. In *EVT'06: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, pages 9–22, Berkeley, CA, USA, 2006. USENIX Association.

[Neu90] Peter G. Neumann. Inside risks: risks in computerized elections. *Commun. ACM*, 33(11):170, 1990.

[PGL00] Gilda Pour, Martin L. Griss, and Michael J. Lutz. The push to make software engineering respectable. *IEEE Computer*, 33(5):35–43, 2000.

[RR06] Brian Randell and Peter Y. A. Ryan. Voting technologies and trust. *IEEE Security and Privacy*, 4(5):50–56, 2006.

[SE03] William L. Scherlis and Jon Eisenberg. IT research, innovation, and e-government. *Commun. ACM*, 46(1):67–68, 2003.

[SL03] Jörgen Svensson and Ronald Leenes. E-voting in europe: Divergent democratic practice. *Information Polity*, 8(1):3–15, 2003.

[SP06] Krishna Sampigethaya and Radha Poovendran. A framework and taxonomy for comparison of electronic voting schemes. *Computers & Security*, 25(2):137–153, 2006.

[TR00] Nicolaus Tideman and Daniel Richardson. Better voting methods through technology: The refinement-manageability trade-off in the single transferable vote. *Public Choice*, 103(1):13–34, April 2000.

2 RESEARCH ACTIVITIES

We can't solve problems by using the same kind of thinking we used when we created them.
Albert Einstein

I hope that posterity will judge me kindly, not only as to the things which I have explained, but also to those which I have intentionally omitted so as to leave to others the pleasure of discovery.
René Descartes

2.1 Director TASS Research Group

From 2002-2005, I was director of the TASS (Theoretical Aspects of Software Systems) research group at NUI Maynooth. In total, under my direction, TASS raised more than 600k euro in research funding, and supported more than 10 postgraduate researchers. The main subgroups of TASS (see section 2.1.2) have grown and evolved since I left the group to work in France 5 years ago.

2.1.1 Why Do We Need Groups such as TASS?

As computing systems have grown in complexity, theoretical computer scientists have built the foundations of the discipline of computing by developing models of computation and methods of analysis applicable to the models. Theoretical computer scientists have driven the development of the science of computing, and make up the majority of the winners of the Turing Award (the "Nobel prize" in the field of computing). Without these scientists the world of computing as we know it would not exist. Although theoretical computer science is mathematical and abstract in spirit — this is the reason why we, as a group, refered to the underlying models as soft systems or software systems rather than hard or concrete — TASS derived its motivation from practical and everyday computation. Its aim was to understand the nature of computation and, as a consequence of this understanding, provide better methods for developing computer systems. With the current focus on the practical relevance of scientific research, it is important to recognize the contributions of theoretical computer science — both the old and the new — to the practice and application of computing across the world.

2.1.2 Research SubGroups: POP and ABC

TASS was structured into two complementary subgroups:

> *Principles of Programming* (POP), led by James Power.
> "The Principles of Programming research group at NUI Maynooth specialises in the static and dynamic analysis of object-oriented programs and programming languages. They exploit a variety of techniques, such as parsing, bytecode analysis, software metrics, meta-modelling and program verification to model software systems in order to increase comprehensibility and reliability. Their work has applications in reverse engineering, program verification and validated forward engineering from design to code. The group's interests extend from software engineering tools and techniques, right through programming language design, down to the implementation of compilers and programming language processors. They have a strong interest in

the formal underpinnings of software technology, and much of their work has links with formal methods in program design and analysis."[2]

Analysis of the Boundaries of Computation (ABC), led by Tom Naugton.

"The study of computer science gives one a firm grasp, through mathematics, of both the theoretical and practical limits of computers, with applications ranging from the design of efficient algorithms, proving the correctness of programs, computer engineering and the Internet, through to artificial intelligence and virtual reality. The subject has an interesting history with pioneers such as Alan Turing (universal computation), Kurt Gödel (limits of computation), John von Neumann (stored-program architecture), Claude Shannon (information theory), and Stephen Cook (computational complexity); and more recently with Tom Head and Len Adleman (DNA computing) and Peter Shor (quantum computing). The ABC group is concerned with exploring the boundaries of computation. In particular, it is concerned with non-standard models of computation, the (fractal) boundary between different levels of algorithmic complexity (with most interest in the boundary between P and NP), analysis of the limits of evolutionary computation and genetic algorithms, and the formulation of a more complete model of complexity (with respect to artificial intelligence and learning)." [3]

2.2 Research Supervision/Direction

2.2.1 PhD Students: Completed

2.2.1.1 McGaley, Margaret — *E-voting: an immature technology in a critical context*. (Submitted: October 2007, accepted: September 2008. Examiners: Prof. Ann Macintosh and Dr. Tom Dowling.). Awarded by the National University of Ireland, Maynooth.

Abstract

"E-voting has been introduced prematurely to national elections in many countries worldwide. There are technical and organizational barriers which must be resolved before the use of e-voting can be recommended in such a critical context. Two fundamental requirements for e-voting systems are in conflict: ballot secrecy and accuracy. We describe the nature and implications of this conflict, and examine the two main categories of proposed solutions: cryptographic voting schemes, and Voter Verified Audit Trails (VVATs). The conflict may permanently rule out the use of remote e-voting for critical elections, especially when one considers that there is no known way to reproduce the enforced privacy of a voting booth outside the supervision of a polling station. We then examine the difficulty faced by governments when they procure Information and Communication Technology (ICT) systems in general, and some mitigation strategies. We go on to describe some legal implications of the introduction of e-voting, which could have serious consequences if not adequately explored, and discuss the evaluation and maintenance of systems. In the final chapters we explore two approaches to the development of requirements for e-voting."

[2] Paraphrased from `http://www.cs.nuim.ie/research/pop/`

[3] Paraphrased from `http://www.cs.nuim.ie/ tnaughton/abc/`

Publication from Margaret McGaley's PhD

[CEB+05] Deirdre Carew, Chris Exton, Jim Buckley, Margaret McGaley, and J.Paul Gibson. Preliminary study to empirically investigate the comprehensibility of requirements specifications. In *Psychology of Programming Interest Group 17th annual workshop (PPIG)*, pages 182–202, University of Sussex, Brighton, UK, 2005.

[GM08] J. Paul Gibson and Margaret McGaley. Verification and maintenance of e-voting systems and standards. In Dan Remenyi, editor, *8th European Conference on e-Government*, pages 283–289. Academic Publishing International, July 2008. ISBN 978-1-906638-09-2.

[McG04] Margaret McGaley. Report on DIMACS Workshop on Electronic Voting — Theory and Practice. Technical report, http://dimacs.rutgers.edu/, 2004.

[McG08] Margaret McGaley. *E-voting: an Immature Technology in a Critical Context*. PhD thesis, Dept. of Computer Science, NUI Maynooth, 2008.

[MG03] Margaret McGaley and J. Paul Gibson. E-voting: a safety critical system. Report NUIM-CS-TR-2003-2, Department of Computer Science, National University of Ireland, Maynooth, 2003.

[MG06] Margaret McGaley and J. Paul Gibson. A critical analysis of the council of europe recommendations on e-voting. In *EVT'06: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, pages 9–22, Berkeley, CA, USA, 2006. USENIX Association.

[MM04] Margaret McGaley and Joe McCarthy. Transparency and e-Voting: Democratic vs. Commercial Interests. In *Electronic Voting in Europe - Technology, Law, Politics and Society*, pages 153 – 163. European Science Foundation, July 2004.

[VM07] Melanie Volkamer and Margaret McGaley. Requirements and evaluation procedures for evoting. In *ARES '07: Proceedings of the The Second International Conference on Availability, Reliability and Security*, pages 895–902, Washington, DC, USA, 2007. IEEE Computer Society.

2.2.1.2 Turlough Neary — *Small Universal Turing Machines*. (Submitted September 2007, accepted November 2007. Examiners: Prof. Maurice Margenstern and Dr. James Power.) Awarded by the National University of Ireland, Maynooth.

I co-supervised this student with Damien Woods, my first PhD student (see below). In Turlough's final year of research on his PhD, I was on sabbatical leave. Consequently, his PhD results are mostly co-authored with Damien.

Abstract

"Numerous results for simple computationally universal systems are presented, with a particular focus on small universal Turing machines. These results are towards finding the simplest universal systems. We add a new aspect to this area by examining trade-offs between the simplicity of universal systems and their time/space computational complexity.

Improving on the earliest results we give the smallest known universal Turing machines that simulate Turing machines in $O(t^2)$ time. They are also the smallest known machines where direct simulation of Turing machines is the technique used to establish their universality. This result gives a new algorithm for small universal Turing machines.

We show that the problem of predicting t steps of the 1D cellular automaton Rule 110 is P-complete. As a corollary we find that the small weakly universal Turing machines of Cook and others run in polynomial time, an exponential improvement on their previously known simulation time overhead.

These results are achieved by improving the cyclic tag system simulation time of Turing machines from exponential to polynomial.

A new form of tag system which we call a bi-tag system is introduced. We prove that bi-tag systems are universal by showing they efficiently simulate Turing machines. We also show that 2-tag systems efficiently simulate Turing machines in polynomial time. As a corollary we find that the small universal Turing machines of Rogozhin, Minsky and others simulate Turing machines in polynomial time. This is an exponential improvement on the previously known simulation time overhead and improves on a forty-year old result.

We present new small polynomial time universal Turing machines with state-symbol pairs of (5, 5), (6, 4), (9, 3) and (15, 2). These machines simulate bi-tag systems and are the smallest known universal Turing machines with 5, 4, 3 and 2-symbols, respectively. The 5-symbol machine uses the same number of instructions (22) as the current smallest known universal Turing machine (Rogozhin's 6-symbol machine).

We give the smallest known weakly universal Turing machines. These machines have state-symbol pairs of (6, 2), (3, 3) and (2, 4). The 3-state and 2-state machines are very close to the minimum possible size for weakly universal machines with 3 and 2 states, respectively."

Publications from Turlough Neary's PhD

[DLM07] Jérôme Olivier Durand-Lose and Maurice Margenstern, editors. *Machines, Computations, and Universality, 5th International Conference, MCU 2007, Orléans, France, September 10-13, 2007, Proceedings*, volume 4664 of *Lecture Notes in Computer Science*. Springer, 2007.

[NW06a] Turlough Neary and Damien Woods. P-completeness of cellular automaton rule 110. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP (1)*, volume 4051 of *Lecture Notes in Computer Science*, pages 132–143. Springer, 2006.

[NW06b] Turlough Neary and Damien Woods. Small fast universal turing machines. *Theor. Comput. Sci.*, 362(1-3):171–195, 2006.

[NW07a] Turlough Neary and Damien Woods. Four small universal turing machines. In Durand-Lose and Margenstern [DLM07], pages 242–254.

[NW07b] Turlough Neary and Damien Woods. Small weakly universal turing machines. *CoRR*, abs/0707.4489, 2007.

[NW09] Turlough Neary and Damien Woods. Four small universal turing machines. *Fundam. Inform.*, 91(1):123–144, 2009.

[WN06a] Damien Woods and Turlough Neary. On the time complexity of 2-tag systems and small universal turing machines. *CoRR*, abs/cs/0612089, 2006.

[WN06b] Damien Woods and Turlough Neary. On the time complexity of 2-tag systems and small universal turing machines. In *FOCS*, pages 439–448. IEEE Computer Society, 2006.

[WN07a] Damien Woods and Turlough Neary. The complexity of small universal turing machines. In S. Barry Cooper, Benedikt Löwe, and Andrea Sorbi, editors, *CiE*, volume 4497 of *Lecture Notes in Computer Science*, pages 791–799. Springer, 2007.

[WN07b] Damien Woods and Turlough Neary. Small semi-weakly universal turing machines. In Durand-Lose and Margenstern [DLM07], pages 303–315.

[WN09a] Damien Woods and Turlough Neary. The complexity of small universal turing machines: A survey. *Theor. Comput. Sci.*, 410(4-5):443–450, 2009.

[WN09b] Damien Woods and Turlough Neary. Small semi-weakly universal turing machines. *Fundam. Inform.*, 91(1):179–195, 2009.

2.2.1.3 Damien Woods — *Computational Complexity of an optical model of computation*. (Submitted November 2004, accepted August 2005. Examiners: Prof. Cris Moore and Prof. Barak Pearlmutter.) Awarded by the National University of Ireland, Maynooth.

Abstract

" We investigate the computational complexity of an optically inspired model of computation. The model is called the continuous space machine and operates in discrete timesteps over a number of two-dimensional complex-valued images of constant size and arbitrary spatial resolution.

We define a number of optically inspired complexity measures and data representations for the model. We show the growth of each complexity measure under each of the model's operations.

We characterise the power of an important discrete restriction of the model. Parallel time on this variant of the model is shown to correspond, within a polynomial, to sequential space on Turing machines, thus verifying the parallel computation thesis. We also give a characterisation of the class NC. As a result the model has computational power equivalent to that of many well-known parallel models. These characterisations give a method to translate parallel algorithms to optical algorithms and facilitate the application of the complexity theory toolbox to optical computers.

Finally we show that another variation on the model is very powerful; illustrating the power of permitting nonuniformity through arbitrary real inputs. "

Publications from Damien Wood's PhD

[WG05a] Damien Woods and J. Paul Gibson. Complexity of continuous space machine operations. In S. Barry Cooper, Benedikt Löwe, and Leen Torenvliet, editors, *CiE*, volume 3526 of *Lecture Notes in Computer Science*, pages 540–551. Springer, 2005.

[WG05b] Damien Woods and J. Paul Gibson. Lower bounds on the computational power of an optical model of computation. In Cristian Calude, Michael J. Dinneen, Gheorghe Paun, Mario J. Pérez-Jiménez, and Grzegorz Rozenberg, editors, *UC*, volume 3699 of *Lecture Notes in Computer Science*, pages 237–250. Springer, 2005.

[WN05] Damien Woods and Thomas J. Naughton. An optical model of computation. *Theor. Comput. Sci.*, 334(1-3):227–258, 2005.

[Woo05] Damien Woods. Upper bounds on the computational power of an optical model of computation. In Xiaotie Deng and Ding-Zhu Du, editors, *ISAAC*, volume 3827 of *Lecture Notes in Computer Science*, pages 777–788. Springer, 2005.

[Woo06] Damien Woods. Optical computing and computational complexity. In Cristian S. Calude, Michael J. Dinneen, Gheorghe Paun, Grzegorz Rozenberg, and Susan Stepney, editors, *UC*, volume 4135 of *Lecture Notes in Computer Science*, pages 27–40. Springer, 2006.

2.2.2 PhD Students: In Progress

Mac Namara, Damien — Limerick Institute of Technology, Ireland — *Electronic Voting: Development of a Dual Vote Architecture*, started 2009, co-supervised with Ken Oakley.

2.2.3 MSc Students: Completed

Walsh, Eamonn, *A secure electronic voting system for academic council elections*, awarded October 2006, Masters of Science in Computer Science, Institute of Technology, Sligo.

Kirk, Mark, *A Client Simulator - And Benchmark Suite - To Test Performance of Multithreaded Architectures in Web Services*, awarded in 2004, Masters of Computer Science, the National University of Ireland, Maynooth.

Phelan, Pat, *Rapid prototyping an educational online game: experimenting with sorting*, awarded in 2004, Masters of Science in Software Engineering, the National University of Ireland, Maynooth.

Hallinan, Stephen, *An Examination of the Use of UML in a Spiral Process with Evolving Requirements and the Subsequent Evaluation of the Design Quality*, awarded in 2004, Masters of Science in Software Engineering, the National University of Ireland, Maynooth.

Laird, Gary, *Globalisation, Localisation and Testing of Visual Studio.NET*, awarded in 2004, Masters of Science in Software Engineering, the National University of Ireland, Maynooth.

Galvin, Valerie, *Evolutionary Prototyping and its application in real-world case study*, awarded in 2004, Masters of Science in Software Engineering, the National University of Ireland, Maynooth.

Shanley, Ray, *Design By Contract In Java*, awarded in 2003, Masters of Science in Software Engineering, the National University of Ireland, Maynooth.

Weir, David, *Simulation and Modelling of Traffic Control Flow*, awarded in 2003, Masters of Computer Science, the National University of Ireland, Maynooth.

Feraille, Matthieu, *Software Engineering Practices At DATACEP (ALTRAN)*, awarded in 2002, Masters of Science in Software Engineering, the National University of Ireland, Maynooth.

Honan, Dermot, *Peer to Peer Computing: An evaluation of the benefits when applied to Content Distribution*, awarded in 2002, Masters of Science in Software Engineering, the National University of Ireland, Maynooth.

Perez, Fran, *The Sequence Pattern*, awarded in 2002, Masters of Science in Software Engineering, the National University of Ireland, Maynooth.

Touzery, Emmanuel, *PCSOFT: managing success*, awarded in 2002, Masters of Science in Software Engineering, the National University of Ireland, Maynooth.

Zou, Jianming, *Improving the Software Development Process at Ericsson*, awarded in 2002, Masters of Science in Software Engineering, the National University of Ireland, Maynooth.

Meagher, Anne, *Automated Student Profiling: A Software Engineering Study of Automated Student Profiling for Teaching Programming*, awarded in 2001, Masters of Science in Software Engineering, the National University of Ireland, Maynooth.

Leacy, Helen, *Animating formal specifications: a JAVA GUI*, awarded in 1999, Masters of Computer Science, the National University of Ireland, Maynooth.

2.3 Theses Examination

2.3.1 PhD

Dubravka Ilic: *Formal reasoning about Dependability in Model-Driven Development*, awarded by Turku Center for Computer Science, 2007.

Mark Hennesy: *A test-driven development strategy for the construction of grammar-based software*, awarded by NUI Maynooth, 2007.

2.3.2 MPhil

Paul Stacey *Peer-to-peer Searching and Sharing of Electronic Documents*, awarded by DIT, Ireland, 2005.

2.3.3 MSc

David Tunney: *var-pi: A language based on the pi-calculus*, awarded by DCU, Ireland, 2004.

Aidan Haran: *Collaborative Computer Personalities in the Game of Chess*, awarded DCU, Ireland, 2002.

2.4 Journals: Reviewing and Editorial Committees

Since 2008, Paul is on the editorial comittee (comité de rédaction) of the *Annals of Telecommunications* (**ISSN: 0003-4347**), published by Springer.
Paul has acted as reviewer for the following journals:

ACM Transactions on Computing Education **ISSN: 1946-6226** .
Annals of Telecommunications **ISSN: 0003-4347**.
Automated Software Engineering **ISSN: 0928-8910**.
Computer Networks (and ISDN Systems) **ISSN: 0169-7552**.
Formal Aspects of Computing **ISSN: 0934-5043**.
International Journal of Foundations of Computer Science (IJFCS) **ISSN: 0129-0541**.
International Journal of Modelling and Simulation **ISSN: 0228-6203**.
Journal of Cultural Heritage, **ISSN: 1296-2074**.
Journal of Systems and Software, **ISSN: 0164-1212**.
Requirements Engineering Journal **ISSN: 0947-3602**.
Science of Computer Programming **ISSN: 0167-6423**.
Software:Practice and Experience, **ISSN: 0038-0644**.

2.5 Conferences and Workshops: Committees and Reviewing

C: (co)chair, **SC:** Session (co)chair, **OC:** organisation committee, **PC:** programme committee, **R:** reviewer —
2011:

Business Modelling and Software Design (BMSD 2011) **(PC, R)**
Conference on e-democracy, e-participation and e-voting (CeDEM 2011) **(R)**
International Conference on Software Engineering Advances (ICSEA11) **(PC, R)**
International Conference on Software and Data Technologies (ICSOFT11) **(PC,R)**
International Conference on Intensive Applications and Services (INTENSIVE 2011) **(PC,R)**
Requirements Engineering for E-voting Systems (RE-Vote11) **(PC,R)**
Special Interest Group in Computer Science Education (SIGCSE 2011) **(R)**
Telecommunications, Networks and Systems (TNS 2011) **(PC,R)**
UML and Formal Methods International workshop (UML&FM 2011) **(PC,R)**

2010:

International Conference on Software Engineering Advances (ICSEA10) **(PC, R)**
International Conference on Software and Data Technologies (ICSOFT10) **(PC,R)**
International Conference on Intensive Applications and Services (INTENSIVE 2010) **(PC,R)**
Special Interest Group in Computer Science Education (SIGCSE 2010) **(R)**
From Research to Teaching Formal Methods - The B Method (TFM-B 2010) **(PC,R)**
Telecommunications, Networks and Systems (TNS 2010) **(PC,R)**

2009:

International Conference on Software Engineering Advances (ICSEA09) **(PC, R)**
International Conference on Software and Data Technologies (ICSOFT09) **(PC,R)**
International Conference on Intensive Applications and Services (INTENSIVE 09) **(PC,R)**
Innovation and Technology in Computer Science Education (ITiCSE 2009) **(R)**
Requirements Engineering for E-voting Systems (RE-Vote09) **(C,PC)**
Special Interest Group in Computer Science Education (SIGCSE 09) **(R)**
Telecommunications, Networks and Systems (TNS 2009) **(PC,R)**

2008:

Innovation and Technology in Computer Science Education (ITiCSE 2008) **(R)**
Special Interest Group in Computer Science Education (SIGCSE 08) **(R)**
Int. Symposium on Leveraging Applications of Formal Methods, Verification & Validation (ISoLA2008) **(PC)**
International Conference on Software Engineering Advances (ICSEA08) **(PC, R)**
Telecommunications, Networks and Systems (TNS 2008) **(PC, R)**
International Conference on Software and Data Technologies (ICSOFT08) **(PC,R)**
African Conference on Research in Computer Science & Applied Mathematics (CARI'08) **(R)**
Workshop on Formal Methods Education and Training (FMET 2008) **(PC)**

2007:

Special Interest Group in Computer Science Education (SIGCSE 07) **(R)**
Innovation and Technology in Computer Science Education (ITiCSE 2007) **(R)**
International Conference on Software Engineering Advances (ICSEA07) **(PC, R)**
International Conference on Software and Data Technologies (ICSOFT07) **(PC,R)**

2006:

Formal Methods Europe (FM(E)06) **(R)**
International Conference on Software Engineering Advances (ICSEA06) **(PC, R)**
International Conference on Software and Data Technologies (ICSOFT06) **(PC,R)**
Information Technology and Telecommunications Doctoral Symposium (IT&T 2006) **(R)**

2004:

Principles and Practice of Programming in Java (PPPJ04) **(C)**
Foundations : Validation & Verification Workshop (Found04(V&V)) **(PC)**

2003:

Feature Interaction Workshop (FIW03) **(PC,R)**
Principles and Practice of Programming in Java (PPPJ03) **(OC,R)**

2002:

Forum on Specification and Design Languages (FDL02: SFP track) **(R)**
Principles and Practice of Programming in Java (PPPJ02) **(R)**
Symposium on Applied Computing (SAC02 (Soft.Eng. Track)) **(SC)**
Artificial Intelligence and Cognitive Science (AICS02) **(R)**

2001:

Irish Workshops in Formal Methods (IWFM01) **(PC,R)**
Intermediate Representation Engineering for the Java Virtual Machine (IRE2001) **(PC,R)**
Formal Methods for Parallel Programming: Theory & Applications (FMPPTA01) **(R)**

2000:

Irish Workshops in Formal Methods (IWFM00) **(C,OC,PC,R)**
Formal Methods for Parallel Programming: Theory & Applications (FMPPTA00) **(R)**

1999:

Formal Methods Europe (FME 1999) **(R)**
Irish Workshops in Formal Methods (IWFM99) **(PC,R)**
Formal Methods for Parallel Programming: Theory & Applications (FMPPTA99) **(R)**

1998:

Formal Methods for Parallel Programming: Theory & Applications (FMPPTA98) **(R)**

2.6 (Inter)national Research Projects: Evaluation and Reviewing

2.6.1 Ireland

Computer science panellist for the Enterprise Ireland *Basic Research Grant* Programme (2003).

2.6.2 European Union

In the last 8 years I have regularly, on average once per year, participated in the EU research framework programme as an evaluator, reviewer, rapporteur and analyst. This usually involves between 1 and 3 weeks of full-time work per year.

The role of evaluator is to examine and evaluate research proposals. I have evaluated many different types of proposal in domains usually related to software engineering (for example: services, open source development, future technologies, grid and cloud computing, architectures, ...). Each proposal requires 6-12 hours of time to evaluate, and then 2-3 hours of meeting time for a consensus to be reached with other evaluators. The process of evaluation helps me to keep up-to-date with much of the recent world-wide research in software engineering, which has a postive effect on my own research and teaching.

The role of reviewer is to provide year-by-year feedback on a research project that has already received funding (after a positive evaluation). This requires about 1 week of work per year — to examine all deliverables, meet with the project members, EU administrators and the other evaluators, and to submit a report with a list of recommendations. My reviewing of the RODIN project helped me achieve an in-depth knowledge of the Event-B language and RODIN platform, and this has led to my use of them in my own research and teaching.

The role of rapporteur is to aid the reviewers to achieve consensus on a proposal, and to draft a consensus report. This task requires the rapporteur to read each proposal and understand the key technical elements; but not to influence the reviewers during the consensus meetings. In general, this task is a little more time consuming than that of an evaluator as it involves additional administration duties. The rapporteur is also responsible for maintaining a high quality in the standard of technical writing that is used in returning results to the team that submitted the proposal.

The role of analyst is to aid the EU in analysing the evaluation process, summarising the results of the process, and making recommendations for future calls.

The specific calls in which I have been involved are listed below:
Evaluator for European FP6, IST priority 2:

Call 2 — IST-2002-2.3.2.3 — *Open development platforms for software and services*
Call 5 — IST-2005-2.5.5 — *Software and Services*

Portfolio Analysis for European FP6, IST priority 2:

Call 5 — IST-2005-2.5.5 — *Software and Services*

Reviewer for RODIN project (European Union, FP6, IST priority 2).
Evaluator/Rapporteur European Union FP7:

ICT objective 1.2 — *Service and Software Architectures, Infrastructures and Engineering*
FET Proactive, FP7 Call 3 — *ICT Forever Yours*
FET Proactive, FP7 Call 4 — *CO-PI: Co-ordinating Communities, Plans and Actions*
FET Proactive, FP7 Call 4 — *TERACOMP: Concurrent Tera-Device Computing*
FP7 ICT Call 5, Objective 1.2 — *Internet of Services, Software and Virtualisation*
FP7 Marie Curie Action Calls — *Mathematics and Engineering*

2.7 Grants Received

CNRS Chercheur Associé 2005 — with Dominique Méry, MOSEL research group, INRIA/LORIA, VandIJuvre-lès-Nancy, *Les méthodes formelles et le problème du transfert de technologie : la nécessité dŠune recherche fondamentale dans la pédagogie du génie logiciel* **(salary from 1st Sept 2005 to 31st August 2006)**

Enterprise Ireland International Collaboration Research Grant (2005) EI/IC/2005/49 — with Rosemary Monahan and Jackie O'Kelly — between Clemson University and NUIM. *Problem-Based-Learning (PBL) - from theory to practice* **(4500 euro)**

IRCSET Embark Initiative (2004-2007) — Turlough Neary, J. Paul Gibson. *The boundaries of complexity hierarchies: maximising problem solving potential by using different models of computation* **(57,150 euro)**.

IRCSET Embark Initiative (2003-2006) — Margaret McGaley, J. Paul Gibson. *Electronic voting: An analysis of the safety critical issues* **(57,150 euro)**.

IRCSET Embark Initiative (2003-2006) — Des Traynor, J. Paul Gibson. *The synthesis and analysis of student profile models in adaptive learning environments for teaching computer programming.* **(57,150 euro)**

IRCSET Embark Initiative (2003-2006) — Ciaran O'Floinn, J. Paul Gibson. *Formalisation of Cryptographic Metrics and its application to emerging techniques.* **(57,150 euro)**

IRCSET Embark Initiative (2002-2005) — Aidan Delaney, J. Paul Gibson, Thomas J. Naughton. *Specification of an abstract operating system running on a single stack push down automaton* **(57,150 euro)**

IRCSET Embark Initiative (2002-2004) — Damien Woods, J. Paul Gibson. *Computational Models and the Turing Limit: An Investigation of the Boundary Between Discrete and Continuous Systems* **(38,100 euro)**

NUI, Maynooth New Researcher Award (2001) — for supporting the costs incurred by my PhD students when attending conferences and workshops, and visiting overseas research laboratories. **(4,000 Irish pounds)**

Enterprise Ireland International Collaboration Research Grant IC/2201/061 (2001) — with Rosemary Monahan and James Power — for the establishment of a formal methods alliance between Clemson University and NUIM. **(6,000 Irish pounds)**

Universite de Metz, Visiting Fellowship (2000) - for collaboration with Dominique Cansell in the application of theorem proving techniques in real world software engineering. **(2,000 Irish pounds)**

Enterprise Ireland Strategic Research Grant SRG/2000/94 (2000-2003) — NUIM-DCU formal methods group collaboration on the *IMPROVE (IMplementing PROtocol Verification for E-commerce) project* — concerned with security protocol specification and verification (working with Baltimore Technologies Ltd). **(70,500 Irish Pounds)**

Enterprise Ireland French Collaboration Grant — with Geoff Hamilton (DCU) (1998 — 2000) — to investigate the formal specification of telephone systems, in collaboration with the Model Group at Loria in Nancy, France. **(1,200 Irish pounds * 3)**

2.8 Invited Talks

Research In Context: A talk in images. Presented at: Postgraduate Research Symposium, Limerick Insitute of Technology (April 2011).

A compositional approach to modelling and formal verification of e-voting systems. Presented at: Department of Information Technology, Limerick Insitute of Technology (April 2010).

A compositional approach to modelling and formal verification of e-voting systems. Presented at: The Computer science department, Namur, Belgium; to the Precise research group (February 2010).

Feature Interactions in a Software Product Line for E-voting. Presented at: The Computer science department, Clemson Univ., SC, USA, to the RSRG research group (September 2009). Co-presenter: Jean-Luc Raffy.

E-voting verfication problems across the world. Presented at: VETO08 (Workshop sur La sécurité Informatique et le Vote ElecTrOnique), CIRM, Marseille Luminy, Université de la Méditerranée (March 2008). Co-presenters: Jean-Luc Raffy and Eric Lallet.

Formal methods — never too young to start. Presented at: "To B or in any Event To B" Seminar/Workshop, Nancy, France (December 2007).

E-voting and the need for rigorous software engineering — the past, present and future. Presented at: B2007, Besancon, France (January 2007).

Trust and security in e-voting systems: the verification problem. Presented at: Workshop on Trustworthy Software, Saarland University, Saarbrücken, Germany (May 2006).

Problem-based learning: the Pablo Picasso Approach. Presented for: The Office of Teaching Effectiveness and Innovation, Clemson Univ., SC, USA (May 2006).

E-voting: software engineering and formal methods. Presented at: The Computer science department, Clemson Univ., SC, USA, to the RSRG research group (May 2006).

Le vote électronique, les methodes formelles et les problèmes complexes dus à la sécurité. Presented to: The MOSEL research group, Nancy, France. (March 2006).

PBL — A computer science viewpoint. Presented at: Project and Problem Based Learning in Higher Education, Galway (June 2003). Co-presenters: Jackie O'Kelly and George Mitchell.

The Hunt For Software Engineers - i'll provide the (silver) bullets if you provide the guns to fire them. Presented at: NUI Maynooth CS Department, Seminar Series (October 2002).

The Complexity of Beauty or the Beauty of Complexity. Presented for: The NUIM Astro-2 (student society) (April 2002).

Fair Objects: Infinity and Beyond. Presented at: Formal Methods Alliance (Clemson University) (January 2002).

An Introduction To Formal Methods. Presented at: Formal Methods Alliance (Clemson University) (January 2002).

Computability, complexity, correctness, and common-sense. Presented at: NUIM Mathematics Seminar (February 2001).

The fractal-like nature of complexity boundaries. Presented at: NUIM-DCU Formal Methods and Security Seminar Series (March 2001).

Software Engineering and Ethics: when code goes bad. Presented for: The NUIM Astro-2 (student society) (March 2001).

The role of computer science in software engineering. Presented at: NUI Maynooth CS Department, Seminar Series (8th May 2000).

Three interesting problems in computer science. Presented at: NUI Maynooth CS Department, Seminar Series (20th November 2000).

Correctness Preserving Transformations for software maintenance (in C++). Presented at: NUI Maynooth CS Department, Seminar Series (25th September 2000). Co-presenters: Prof. Brian Malloy and Dr T Dowling.

Stability issues in formal OO requirements models. Presented at: NUI Maynooth CS Department, Seminar Series (1st November 1999).

2.9 Research Group Membership

AVERSE (Administration, Validation et sEcurité des Réseaux et SErvices) as part of L'Unité Mixte de Recherche SAMOVAR (Services répartis, Architectures, MOdélisation, Validation, Administration des Réseaux) UMR 5157 INT CNRS Research Laboratory.
(see: `http://samovar.it-sudparis.eu/equipes/eq_averse.php`)
MOSEL, INRIA/LORIA, Nancy, France.
(see: `http://www.loria.fr/equipes/model/`)
FMA (Formal Methods Alliance), Clemson, USA.
(see: `http://www.cs.clemson.edu/~steve/FMG/group.html`)

2.10 Publications

(Co)-Editor Proceedings

[1] Isabelle Perseil and J. Paul Gibson, editors. *Fourth IEEE International workshop UML and Formal Methods (UML&FM2011)*, Limerick, Ireland, june 2011. IEEE.

[2] J. Paul Gibson and Doug Jones, editors. *First International Workshop on Requirements Engineering for e-Voting Systems (RE-VOTE09)*, Atlanta,GA,USA, August 2009. IEEE.

[3] J. Paul Gibson, James Power, and John Waldron, editors. *PPPJ '04: Proceedings of the 3rd international symposium on Principles and practice of programming in Java*, Las Vegas, Nevada, 2004. Trinity College Dublin. General Chair-John Waldron.

[4] David C. Rine, James F. Power, and J. Paul Gibson. ACM SAC2002 software engineering: theory and applications (SETA) track description. In *ACM Symposium on Applied Computing (SAC 2002)*, pages 969–970, Madrid, Spain, 2002. ACM.

[5] David Sinclair and J. Paul Gibson, editors. *4th Irish Workshop on Formal Methods (IWFM 2000)*, Electronic Workshops in Computing, Maynooth, Ireland, July 2000. BCS.

International Journal Publications

[6] J. Paul Gibson, Eric Lallet, and Jean-Luc Raffy. Formal object oriented development of a voting system test oracle. *Innovations in Systems and Software Engineering (Special issue UML-FM11)*, 2011. To appear.

[7] Damien MacNamara, Ted Scully, Francis Carmody, Ken Oakley, Elizabeth Quane, and J. Paul Gibson. Dual vote: A non-intrusive evoting interface. *International Journal of Computer Information Systems and Industrial Management Applications(IJCISIM)*, 2011. To appear.

[8] J. Paul Gibson, Eric Lallet, and Jean-Luc Raffy. How do I know if my design is correct? *Electronic Notes in Theoretical Computer Science (ENTCS)*, 2011. To appear.

[9] J. Paul Gibson. Formal methods - never too young to start. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 2011. To appear.

[10] J. Paul Gibson. Software reuse and plagiarism: A code of practice. *SIGCSE Bull.*, 41(3):55–59, 2009.

[11] Damien Woods and J. Paul Gibson. Lower bounds on the computational power of an optical model of computation. *Natural Computing*, 7(1):95–108, 2008.

[12] Dominique Cansell, J. Paul Gibson, and Dominique Méry. Refinement: A constructive approach to formal software design for a secure e-voting interface. *Electronic Notes in Theoretical Computer Science*, 183:39–55, 2007.

[13] Jackie O'Kelly and J. Paul Gibson. Robocode & problem-based learning: a non-prescriptive approach to teaching programming. *SIGCSE Bull.*, 38(3):217–221, 2006.

[14] Des Traynor and J. Paul Gibson. Synthesis and analysis of automatic assessment methods in CS1: generating intelligent MCQs. *SIGCSE Bull.*, 37(1):495–499, 2005.

[15] J. Paul Gibson and J.A. Lynch. Applying formal object oriented design principles to Smalltalk-80. *British Telecom Technology Journal*, 3, July 1989.

Inter. Peer-Reviewed Conference Proceedings & Book Chapters

[16] J. Paul Gibson, Damien MacNamara, and Ken Oakley. Just like paper and the 3-colour protocol: a voting interface requirements engineering case study. In *Proceedings of RE-Vote 2011*, Trento, Italy, august 2011. IEEE.

[17] Damien MacNamara, Ted Scully, J. Paul Gibson, Francis Carmody, Ken Oakley, and Elizabeth Quane. Dualvote: Addressing usability and verifiability issues in electronic voting systems. In *2011 Conference for E-Democracy and Open Government (CeDEM11)*, Danube University, Krems, May 2011. Edition Danube University.

[18] Damien MacNamara, Francis Carmody, Ted Scully, Ken Oakley, Elizabeth Quane, and J. Paul Gibson. Dual vote: A novel user interface for e-voting systems. In *IADIS International Conference on Interfaces and Human Computer Interaction 2010*, Freiburg, Germany, 28 - 30 July 2010, 2010. IADIS.

[19] Kevin Casey and J. Paul Gibson. (m)Oodles of Data Mining Moodle to understand Student Behaviour. In Fiona O'Riordan, Fergus Toolan, Rosario Hernandez, Robbie Smyth, Brett Becker, Kevin Casey, David Lillis, Geraldine McGing, Majella Mulhall, and Kay O'Sullivan, editors, *ICEP 10 Conference Papers: Engaging Pedagogy*, pages 61–71, Maynooth, Ireland, December 2010. Griffith College Dublin.

[20] J. Paul Gibson, Eric Lallet, and Jean-Luc Raffy. Engineering a distributed e-voting system architecture: Meeting critical requirements. In Holger Giese, editor, *Architecting Critical Systems, First International Symposium, ISARCS 2010, Prague, Czech Republic, June 23-25, 2010, Proceedings*, volume 6150 of *Lecture Notes in Computer Science*, pages 89–108. Springer, 2010.

[21] J. Paul Gibson, Eric Lallet, and Jean-Luc Raffy. Sculpturing Event-B models with Rodin: "holes and lumps" in teaching refinement through problem-based learning. In *From Research to Teaching Formal Methods - The B Method (TFM B'2009)*, pages 7–21, Nantes, France, 2009. APCB.

[22] J. Paul Gibson, Eric Lallet, and Jean-Luc Raffy. Feature interactions in a software product line for e-voting. In Nakamura and Reiff-Marganiec, editors, *Feature Interactions in Software and Communication Systems X*, pages 91–106, Lisbon, Portugal, June 2009. IOS Press.

[23] J. Paul Gibson. Software reuse and plagiarism: A code of practice. In *14th ACM SIGCSE Annual Conference on Innovation and Technology in Computer Science Education(ITiCSE 2009)*, pages 55–59, Paris, France, July 2009. ACM.

[24] J. Paul Gibson. Challenging the lecturer: Learning from the teacher's mistakes. In Fiona O'Riordan, Fergus Toolan, Rosario Hernandez, Robbie Smyth, Brett Becker, Kevin Casey, David Lillis, Geraldine McGing, Majella Mulhall, and Kay O'Sullivan, editors, *ICEP 09 Conference Papers: Engaging Pedagogy*, pages 61–71, Dublin, Ireland, November 2009. Griffith College Dublin.

[25] J. Paul Gibson. Weaving a formal methods education with problem-based learning. In T. Margaria and B. Steffen, editors, *3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*, volume 17 of *Communications in Computer and Information Science (CCIS)*, pages 460–472, Porto Sani, Greece, October 2008. Springer-Verlag, Berlin Heidelberg.

[26] J. Paul Gibson, Eric Lallet, and Jean-Luc Raffy. Analysis of a distributed e-voting system architecture against quality of service requirements. In Herwig Mannaert, Tadashi Ohta, Cosmin Dini, and Robert Pellerin, editors, *The Third International Conference on Software Engineering Advances (ICSEA 2008)*, pages 58–64, Sliema, Malta, October 2008. IEEE Computer Society.

[27] J. Paul Gibson and Margaret McGaley. Verification and maintenance of e-voting systems and standards. In Dan Remenyi, editor, *8th European Conference on e-Government*, pages 283–289, Lausanne, Switzerland, July 2008. Academic Publishing International.

[28] J. Paul Gibson, Eric Lallet, and Jean-Luc Raffy. How do I know if my design is correct? In Zoltan Istenes, editor, *Formal Methods in Computer Science Education (FORMED 2008)*, pages 61–70, Budapest, Hungary, March 2008. Accepted for publication in ENTCS.

[29] J. Paul Gibson. Formal methods — never too young to start. In Zoltan Istenes, editor, *Formal Methods in Computer Science Education (FORMED 2008)*, pages 151–160, Budapest, Hungary, March 2008. Accepted for publication in ENTCS.

[30] J. Paul Gibson. E-voting and the need for rigorous software engineering — the past, present and future. In Jacques Julliand and Olga Kouchnarenko, editors, *B 2007: Formal Specification and Development in B, 7th International Conference of B Users*, volume 4355 of *Lecture Notes in Computer Science*, page 1, Besançon, France, 2007. Springer.

[31] Dominique Cansell, J. Paul Gibson, and Dominique Méry. Formal verification of tamper-evident storage for e-voting. In Mike Hinchey and Tiziana Margaria, editors, *Fifth IEEE International Conference on Software Engineering and Formal Methods (SEFM 2007)*, pages 329–338, London, England, UK, 2007. IEEE Computer Society.

[32] Dominique Cansell, J. Paul Gibson, and Dominique Méry. Refinement: A constructive approach to formal software design for a secure e-voting interface. In A. Cerone and P. Curzon, editors, *Formal Methods for Interactive Systems (FMIS 2006)*, Macau SAR China, October 2006.

[33] Margaret McGaley and J. Paul Gibson. A critical analysis of the council of europe recommendations on e-voting. In *EVT'06: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, pages 9–22, Berkeley, CA, USA, 2006. USENIX Association.

[34] Jackie O'Kelly and J. Paul Gibson. Robocode & problem-based learning: a non-prescriptive approach to teaching programming. In Renzo Davoli, Michael Goldweber, and Paola Salomoni, editors, *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE 2006*, pages 217–221, Bologna, Italy, 2006. ACM. Also published in ACM SIGCSE Bulletin.

[35] Des Traynor, Susan Bergin, and J. Paul Gibson. Automated assessment in CS1. In *ACE '06: Proceedings of the 8th Austalian conference on computing education*, pages 223–228, Darlinghurst, Australia, 2006. Australian Computer Society, Inc.

[36] Jackie O'Kelly and J. Paul Gibson. Software engineering as a model of understanding for learning and problem solving. In *ICER '05: Proceedings of the 2005 international workshop on Computing education research*, pages 87–97, New York, NY, USA, 2005. ACM.

[37] Damien Woods and J. Paul Gibson. Lower bounds on the computational power of an optical model of computation. In Cristian Calude, Michael J. Dinneen, Gheorghe Paun, Mario J. Pérez-Jiménez, and Grzegorz Rozenberg, editors, *4th International Conference on Unconventional Computation (UC2005)*, volume 3699 of *Lecture Notes in Computer Science*, pages 237–250, Sevilla, Spain, 2005. Springer.

[38] Deirdre Carew, Chris Exton, Jim Buckley, Margaret McGaley, and J. Paul Gibson. Preliminary study to empirically investigate the comprehensibility of requirements specifications. In P. Romero, J. Good, E. Acosta Chaparro, and S. Bryant, editors, *Psychology of Programming Interest Group 17th annual workshop (PPIG 2005)*, pages 182–202, University of Sussex, Brighton, UK, 2005.

[39] Damien Woods and J. Paul Gibson. Complexity of continuous space machine operations. In S. Barry Cooper, Benedikt Löwe, and Leen Torenvliet, editors, *New Computational Paradigms, First Conference on Computability in Europe CiE2005*, volume 3526 of *Lecture Notes in Computer Science*, pages 540–551, Amsterdam, The Netherlands, 2005. Springer.

[40] Jackie O'Kelly and J. Paul Gibson. PBL: Year one analysis — interpretation and validation. In *PBL In Context — Bridging work and Education*, Lahti, Finland, 2005.

[41] Stephen Hallinan and J. Paul Gibson. A graduate's role in technology transfer: From requirements to design with UML. In Peter Kokol, editor, *IASTED International Conference on Software Engineering, part of the 23rd Multi-Conference on Applied Informatics*, pages 94–99, Innsbruck, Austria, 2005. IASTED/ACTA Press.

[42] Des Traynor and J. Paul Gibson. Synthesis and analysis of automatic assessment methods in CS1: generating intelligent MCQs. In Wanda Dann, Thomas L. Naps, Paul T. Tymann, and Doug Baldwin, editors, *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2005)*, pages 495–499, St. Louis, Missouri, USA, 2005. ACM. Also published in ACM SIGCSE Bulletin.

[43] Des Traynor and J. Paul Gibson. Implementing cognitive modelling in CS education: Aligning theory and practice of learning to program. In Kinshuk, Demetrios G. Sampson, and Pedro T. Isaías, editors, *Cognition and Exploratory Learning in Digital Age CELDA 2004*, pages 533–536, Lisbon, Portugal, 2004. IADIS.

[44] Des Traynor and J. Paul Gibson. Towards the development of a cognitive model of programming: a software engineering proposal. In E. Dunican and T.R.G. Green, editors, *Psychology of Programming Interest Group 16th annual workshop (PPIG 2004)*, pages 79–85, 2004.

[45] J. Paul Gibson. A noughts and crosses Java applet to teach programming to primary school children. In James F. Power and John Waldron, editors, *Proceedings of the 2nd International Symposium on Principles and Practice of Programming in*

Java (PPPJ 2003), volume 42 of *ACM International Conference Proceeding Series*, pages 85–88, Kilkenny City, Ireland, 2003. ACM.

[46] Edward B. Duffy, Brian A. Malloy, and J. Paul Gibson. Applying the decorator pattern for profiling object-oriented software. In *11th International Workshop on Program Comprehension (IWPC 2003)*, pages 84–93, Portland, Oregon, USA, 2003. IEEE Computer Society.

[47] Peter J. Clarke, Brian A. Malloy, and J. Paul Gibson. Using a taxonomy tool to identify changes in OO software. In Gerardo Canfora, Mark van den Brand, and Tibor Gyimothy, editors, *7th European Conference on Software Maintenance and Reengineering CSMR 2003*, pages 213–222, Benevento, Italy, 2003. IEEE Computer Society.

[48] J. Paul Gibson. Formal requirements engineering: Learning from the students. In Doug Grant, editor, *12th Australian Software Engineering Conference (ASWEC 2000)*, pages 171–180, Canberra, Australia, 2000. IEEE Computer Society.

[49] David Sinclair, James F. Power, J. Paul Gibson, David Gray, and Geoff Hamilton. Specifying and verifying IP with linear logic. In Ten-Hwang Lai, editor, *ICDCS Workshop on Distributed System Validation and Verification*, pages E104–E110, Taiwan, ROC, 2000.

[50] J. Paul Gibson, Geoff Hamilton, and Dominique Méry. A taxonomy for triggered interactions using fair object semantics. In Muffy Calder and Evan H. Magill, editors, *Feature Interactions in Telecommunications and Software Systems VI (FIW 2000)*, pages 193–209, Glasgow, Scotland, UK, 2000. IOS Press.

[51] Geoff Hamilton, J. Paul Gibson, and Dominique Méry. Composing fair objects. In Fouchal and Lee, editors, *International Conference on Software Engineering Applied to Networking and Parallel/Distributed Computing (SNPD '00)*, pages 225–233, Reims, France, May 2000.

[52] J. Paul Gibson, Thomas F. Dowling, and Brian A. Malloy. The application of correctness preserving transformations to software maintenance. In *ICSM '00: Proceedings of the International Conference on Software Maintenance (ICSM'00)*, pages 108–119, Washington, DC, USA, 2000. IEEE Computer Society.

[53] J. Paul Gibson and Dominique Méry. Formal modelling of services for getting a better understanding of the feature interaction problem. In Dines Bjørner, Manfred Broy, and Alexandre V. Zamulin, editors, *PSI '99: Proceedings of the Third International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, volume 1755 of *Lecture Notes in Computer Science*, pages 155–179, Akademgorodok, Novosibirsk, Russia, 1999. Springer.

[54] J. Paul Gibson. Formal object oriented requirements: simulation, validation and verification. In Helena Szczerbicka, editor, *Modelling and Simulation: A tool for the next millenium ESM99*, volume II, pages 103–111, Warsaw, Poland, June 1999. Society for Computer Simulation International (SCS).

[55] J. Paul Gibson, Dominique Méry, and Yassine Mokhtari. Animating formal specifications - a telephone simulation case study. In Helena Szczerbicka, editor, *Modelling and Simulation: A tool for the next millenium ESM99*, volume II, pages 139–146, Warsaw, Poland, June 1999. Society for Computer Simulation International (SCS).

[56] J. Paul Gibson, Geoff Hamilton, and Dominique Méry. Integration problems in telephone feature requirements. In Keijiro Araki, Andy Galloway, and Kenji Taguchi, editors, *Integrated Formal Methods, Proceedings of the 1st International Conference on Integrated Formal Methods (IFM 99)*, pages 129–148, York, UK, June 1999. Springer.

[57] J. Paul Gibson and Dominique Méry. Fair objects. In H.S.M. Zedan and Antonio Cau, editors, *Object-oriented technology and computing systems re-engineering*, pages 122–140, Chichester, USA, 1999. Horwood Publishing, Ltd.

[58] David Gray, Geoff Hamilton, David Sinclair, J. Paul Gibson, and James F. Power. Four logics and a protocol. In Andrew Butterfield and Klemens Haegele, editors, *3rd Irish Workshop on Formal Methods (IWFM 1999)*, Electronic Workshops in Computing, Galway, Ireland, 1999. BCS.

[59] J. Paul Gibson. Towards a feature interaction algebra. In Kristofer Kimbler and Wiet Bouma, editors, *Feature Interactions in Telecommunications and Software Systems V(FIW 1998)*, pages 217–231, Malmö, Sweden, 1998. IOS Press.

[60] J. Paul Gibson and Dominique Méry. Teaching formal methods: Lessons to learn. In Sharon Flynn and Andrew Butterfield, editors, *2nd Irish Workshop on Formal Methods (IWFM 1998)*, Electronic Workshops in Computing, Cork, Ireland, July 1998. BCS.

[61] J. Paul Gibson and Dominique Méry. Always and eventually in object requirements. In A.S Evans, editor, *Second Workshop on Rigorous Object Oriented Methods (ROOM 2)*, Bradford, West Yorkshire, UK, May 1998.

[62] J. Paul Gibson and Dominique Méry. Telephone feature verification: Translating SDL to TLA+. In Ana R. Cavalli and Amardeo Sarma, editors, *SDL '97 Time for Testing, SDL, MSC and Trends — 8th International SDL Forum*, pages 103–118, Evry, France, September 1997. Elsevier.

[63] J. Paul Gibson. Feature requirements models: Understanding interactions. In Petre Dini, Raouf Boutaba, and Luigi Logrippo, editors, *Feature Interactions in Telecommunications Networks IV, (FIW 1997)*, pages 46–60, Montréal, Canada, June 1997. IOS Press.

[64] J. Paul Gibson, Bruno Mermet, and Dominique Méry. Feature interactions: A mixed semantic model approach. In Henry McGloughlin and Gerard O'Regan, editors, *1st Irish Workshop on Formal Methods (IWFM 1997)*, Electronic Workshops in Computing, Dublin, Ireland, July 1997. BCS.

[65] J. Paul Gibson and Dominique Méry. A unifying model for specification and design. In Galmiche, Bashoun, Fiadero, and Yonezawa, editors, *Proceedings of the Workshop on Proof Theory of Concurrent Object Oriented Programming*, Linz (Austria), July 1996.

Technical Reports

[66] Jackie O'Kelly, Rosemary Monahan, J. Paul Gibson, and Stephen Brown. Enhancing skills transfer through problem-based learning. Report NUIM-CS-TR-2005-13, Department of Computer Science, National University of Ireland, Maynooth., 2005.

[67] J. Paul Gibson. E-voting requirements modelling: An algebraic specification approach (with cafeobj). Report NUIM-CS-TR-2005-14, Department of Computer Science, National University of Ireland, Maynooth., 2005.

[68] J. Paul Gibson. Software reuse in final year projects: A code of practice. Report NUIM-CS-TR-2003-12, Department of Computer Science, National University of Ireland, Maynooth., 2003.

[69] Margaret McGaley and J. Paul Gibson. E-voting: a safety critical system. Report NUIM-CS-TR-2003-2, Department of Computer Science, National University of Ireland, Maynooth., 2003.

[70] J. Paul Gibson. Formal requirements models: simulation, validation and verification. Report NUIM-CS-TR-2001-2, Department of Computer Science, National University of Ireland, Maynooth., 2001.

[71] J. Paul Gibson. On the relationship between computational models and scientific theories. Report NUIM-CS-TR-2001-5, Department of Computer Science, National University of Ireland, Maynooth., 2001.

[72] Damien Woods, Thomas J. Naughton, and J. Paul Gibson. Analog recurrent neural network simulation, O(log n) unordered search, and bitonic sort with an optically-inspired model of computation. Report NUIM-CS-TR-2001-6, Department of Computer Science, National University of Ireland, Maynooth, 2001.

[73] J. Paul Gibson. An OO requirements capture and analysis environment. Technical Report CRIN-98-R-010, Centre de Recherche en Informatique de Nancy (CRIN), January 1998.

[74] J. Paul Gibson and Yassine Mokhtari. POTS: An OO LOTOS specification. Technical Report CRIN-98-R-013, Centre de Recherche en Informatique de Nancy (CRIN), January 1998.

[75] J. Paul Gibson, Dominique Cansell, Bruno Mermet, and Dominique Méry. Spécification de services dans une logique temporelle compositionnelle: Rapport de fin du lot1 du marché. Technical Report no961B 1B CNET-CNRS-CRIN, Centre de Recherche en Informatique de Nancy (CRIN), 1998.

[76] J. Paul Gibson and Dominique Méry. Formal methods for concurrency parallelism and distribution. Rapport Interne CRIN-96-R-378, Centre de Recherche en Informatique de Nancy (CRIN), 1995. Published in ERCIM News *Software Quality* (23).

[77] J. Paul Gibson. *Formal Object Oriented Development of Software Systems Using LOTOS*. Thesis CSM-114, Stirling University, August 1993.

[78] J. Paul Gibson. Formal Object Based Design in LOTOS. Technical Report Computer Science: TR-113, University of Stirling, 1993.

[79] J. Paul Gibson. A LOTOS-Based Approach to Neural Network Specification. Technical Report Computer Science: TR-112, University of Stirling, 1993.

3 RESEARCH CONTRIBUTIONS: THEORETICAL, APPLIED AND EDUCATIONAL — CONTRIBUTIONS DE RECHERCHE : THÉORIQUES, APPLIQUÉES ET PÉDAGOGIQUES

If your contribution has been vital there will always be somebody to pick up where you left off, and that will be your claim to immortality.
Walter Gropius

Etre homme, c'est précisément être responsable. C'est sentir, en posant sa pierre, que l'on contribue à bâtir le monde.
Antoine de Saint-Exupéry

3.1 Software Engineering Fundamental Theory: Feature Composition — Théorie fondamentale du génie logiciel : Composition des services

The feature interaction problem is stated simply, and informally, as follows: A *feature interaction* is a situation in which system behaviour (specified as some set of features) does not as a whole satisfy each of its component features individually.

Feature interaction is a difficult problem made more difficult by the fact that feature combinations cannot be fully understood when individual features are themselves not understood. Formal requirements modeling is the triangular process of gaining understanding, recording understanding in a formal model, and validating the model. By formally developing feature requirements models we can approach the problem of combining features with much more confidence. The problem is still as difficult, but at least it is now well defined. We argue that improving understanding — through foundational research — is the key to interaction avoidance, detection and resolution.

Le problème de l'interaction des services (*features*) est établi simplement et informellement comme suit : une *interaction* est une situation dans laquelle le system behaviour (défini comme un ensemble de (*features*) ne satisfait pas, en tant que "tout", chacune de ses *features* qui le composent individuellement.

La *feature interaction* est un problème difficile rendu encore plus difficile par le fait que les combinaisons ne peuvent pas être entièrement comprises lorsque les features elles-mêmes ne sont pas comprises. La modélisation des besoins formels est le procédé triangulaire qui consiste à gagner la compréhension, enregistrer la compréhension dans un modèle formel, et valider le modèle. En développant de façon formelle les modèles d'exigences en *feature*, nous pouvons approcher le problème en combinant des *features* avec beaucoup plus de confiance. Le problème reste tout autant difficile mais, au moins, il est maintenant bien défini. Nous affirmons qu'améliorer la compréhension – par le biais d'une recherche de fondements — est la clé pour éviter l'interaction, pour détecter et pour résoudre.

3.1.1 Features and requirements engineering — Services et ingénierie des besoins

Many of the problems which arise when features combine (i.e. *feature interactions*) are due to *badly* developed requirements models for individual features. With sufficiently *good* requirements

Un grand nombre des problèmes qui surviennent lorsque les *features* se combinent sont dus au mauvais développement des modèles des besoins pour *features* individuelles. Avec des modèles de besoins suffisam-

models, in which each feature is formally modelled and validated against customer understanding, the feature interaction problem is much more tractable.

In [Gib97] we analyse some of the *standard* feature interactions within telephone systems and show that, in most cases, the notion of interaction is used to signify that the requirements are not fully understood, properly recorded or rigorously validated. In contrast, we then show how *good* requirements models could resolve the problems that arise when combining features. An interaction is said to occur if and only if requirements are contradictory. The problem which this paper addresses is how to avoid, detect and resolve such contradictions during requirements development.

Features are observable behaviour and are therefore a requirements specification problem [Zav93]. Most feature interaction problems can be (and should be) resolved at the requirements capture stage of development. If there are no problems in the requirements specification then problems during the design and implementation will arise only through errors in the refinement process. Certainly the feature interaction problem is more prone to the introduction of such errors because of the highly concurrent and distributed nature of the underlying implementation domain, but this is for consideration *after* each individual feature's requirements have been modelled and validated; otherwise it will not be easy to identify the source of the interaction. Features are requirements modules *and* the units of incrementation as systems evolve. A telecom system is a set of features.

A feature interaction occurs in a system whose complete behaviour does not satisfy the separate specifications of all its features. Having features as the incremental units of development is the source of many of the problems:

Complexity explosion — Potential feature interactions increase exponentially with the number of features

ment bons, dans lequel chaque *feature* est formellement modélisée et validée face à la compréhension du client, l'interaction est beaucoup plus soluble.

Dans [Gib97], nous analysons certaines des interactions standards au sein de systèmes de téléphone et nous montrons que, dans la plupart des cas, la notion d'interaction est utilisée pour signifier que les exigences ne sont pas entièrement compriser, proprement enregistrées ou rigoureusement validées. Par opposition, nous montrons alors comment de bons modèles de besoins peuvent résoudre les problèmes qui surviennent quand on combine les *features*. Une interaction est supposée apparaître si, et seulement si, les besoins sont contradictoires. Le problème abordé par cet essai est de savoir comment éviter, détecter et résoudre de telles contradictions pendant l'ingénierie des besoins.

Features sont des comportements observable et, par conséquent, sont un "requirements specification problem" [Zav93]. La plupart des problèmes d'*interaction* peuvent être (et devraient être) résolus pendant l'ingénierie des besoins. Si aucun problème n'apparaît au niveau de la spécification de besoins, alors les problèmes pendant la conception et l'implémentation surviendront seulement au travers d'erreurs commises au cours du processus de raffinement. Il est certain que ce problème est plus enclin à l'introduction de telles erreurs en raison de la nature hautement concurrente et répartie de l'architecture ; mais ceci est sous considération après que le comportement de chaque *feature* a été modélisé et validé ; sinon il ne sera pas aisé d'identifier la source de l'interaction. *Features* sont des composants des besoins et des unités d'incrémentation alors que les systèmes évoluent. Un système de télécommunication est un ensemble de *features*.

Une interaction apparaît dans un système où le comportement complet ne satisfait pas aux spécifications séparées de toutes ses *features*. Posséder des *features* comme unités incrémentielles de développement constitue la source de nos problèmes :

L'explosion combinatoire — les interactions potentielles augmentent de façon exponentielle avec le nombre de *features*

in the system.

Chaotic Information Structure In Sequential Development Strategies — The arbitrary sequential ordering of feature development is what drives the internal structure of the resulting system. As each new feature is added the feature must include details of how it is to interact with all the features already in the system. Consequently, to understand the behaviour of one feature, it is necessary to examine the specification of all the features in the system. All conceptual integrity is lost since the distribution of knowledge is chaotic. The arbitrary ordering of feature development has a large effect on the internal system structure.

Assumption Problem — Already developed features often rely on assumptions which are no longer true when later features are conceived. Consequently, features may rely on contradictory assumptions.

Independent Development — Traditional approaches require a new feature developer to consider how the feature operates with all others already on the system. Consequently, we cannot concurrently develop new features: since how the new features work together will not be considered by either of the two independent feature developers. We want to have an incremental approach in which the developers do not need to know anything about the other features in the system. In our approach, it is the system designers who must resolve the integration problems: integration methods arise from an analysis of the features to be integrated. Formal requirements of individual features are required for the integration process to be verified.

Interface Problem — User controls on traditional phones are very limited and hence the input signals become polymorphic. This is a major problem in requirements specifications as it can lead to the introduction of ambiguities in systems of features. Formal requirements models make explicit the mapping between abstract and concrete actions and our systems can be automatically verified to ensure an absence of ambiguity that could lead to interactions.

Invalid Plain Old Telephone Service (POTS) As-

dans le système.

L'ordre séquentiel arbitraire du développement est ce qui conduit la structure interne du système résultant. Comme chaque nouvelle *feature* est ajoutée, la *feature* doit inclure les détails de la manière dont elle est supposée interagir avec les autres features déjà présentes dans le système. Par conséquent, pour comprendre le comportement d'une *feature*, il est nécessaire d'examiner la spécification des toutes les *features* au sein du système. Toute intégrité conceptuelle est perdue car la distribution de connaissances est chaotique. L'ordre séquentiel arbitraire du développement possède un large effet sur la structure du système interne.

Problème de supposition — des *features* déjà développées dépendent souvent de suppositions qui ne sont plus vraies lorsque des *features* ultérieures sont conçues. Par conséquent, des *features* peuvent dépendre de suppositions contradictoires.

Développement indépendant — les approches traditionnelles exigent que le développeur d'une nouvelle *feature* considère comment la *feature* opère avec toutes les autres déjà sur le système. Par conséquent, comme la façon dont les nouvelles *features* travaillent ensemble ne sera considérée par aucun des deux développeurs de *feature* indépendants, il nous est impossible actuellement de développer de nouvelles *features*. Nous voulons avoir une approche incrémentielle dans laquelle les développeurs n'ont pas besoin de tout savoir sur les autres *features* du système. Dans notre approche, ce sont les architectes du système qui doivent résoudre les problèmes d'intégration ; en effet, les méthodes d'intégration émanent d'une analyse des *features* devant être intégrées. Les besoins formelles de chaque *feature* sont requis pour que le processus d'intégration soit vérifié.

Problème d'interface — les contrôles d'utilisateurs sur téléphones traditionnels sont très limités et ainsi les signaux d'entrée deviennent polymorphiques. Ceci est un problème majeur dans les spécifications de besoins car cela peut conduire à l'introduction d'ambiguïtés dans les systèmes de *features*. Les modèles de besoins formels rendent explicite la projection entre des actions abstraites et concrètes, et notre système peut être automatiquement vérifié pour assurer l'absence d'ambiguïté pouvant conduire à des interactions.

Suppositions invalides des Plain Old Telephone Service

sumptions — Phone systems have changed dramatically over the past ten years. Many people (including feature developers) are not aware of the underlying complexity in the concrete system and, as a way of simplifying the problem, often make incorrect assumptions based on their knowledge of the plain old telephone service.

(POTS) — les systèmes de téléphone ont considérablement changé au cours des dix dernières années. De nombreuses personnes (incluant les développeurs de *features*) ne sont pas conscients de la complexité sous-jacente dans le système concret et, afin de simplifier le problème, font souvent des suppositions fondées sur leur connaissance du simple et vieux système de téléphonie.

The feature interaction problem is difficult: our research shows that having formal requirements models makes it manageable[Gib97]. The main contribution of this paper was to identify a number of guiding principles:

Use Strong Typing — The advantages of typing in all forms of development are well known. Types are not needed in correct systems but they do help to create correct systems. A simple telephone example is that of the concept of a telephone number. Clearly, telephone numbers are polymorphic within a concrete subtyping hierarchy. Without types, it is difficult to handle the different natures of telephone numbers at the requirements stage.

Use Types as Behaviours or Roles — Here we impose the object oriented principle of classification. The class is used as the fundamental unit of behaviour (and as the means of typing these units). Thus, every feature is a class which plays a specific role. The class hierarchy provides a behavioural benchmark for categorising features.

Use Invariants — Invariants are used to define relationships between components of a system that must be true during the lifetime of the system. They are a well understood, formal means of specifying requirements in a compositional manner. Every non-trivial component (of a (sub)system), i.e. one with its own components, has an associated invariant and there is one invariant between all components (of a (sub)system). Invariants are the logical glue for putting together systems from component parts.

Avoid ambiguity in naming of actions — The practice of polymorphic actions arises from the minimilist interface provided by most telephones. For example, a *flash hook* action can signal different things to different features. If these features are requested at the same time then the meaning of a *flash hook* may be ambiguous. Whenever possible try to maintain a clear distinction between *abstract actions/signals* in the requirements model and *concrete actions/signals* in the implementation model. Finding a *correct* mapping between abstract and concrete action names is a design problem and not inherently a feature interaction problem; although, the limited domain of concrete actions may make this more difficult (or impossible) in the case of feature development.

Features should be explicit not implicit — Typically, features appear in formal specifications only as implicit, derivable properties of the total system. We can verify that a system complies to the requirments of a feature but there is no compositional means of removing the feature and examining it as a single identity. Given a logical landscape as our semantic basis would permit such a compostional view since the properties required of a feature are exactly its specification. However, in such a logical approach, the actual development of features is much more complicated. We believe that we should have an *explicit* compositional approach in the same spirit as the logical method, whilst avoiding the synthesis and analysis problems that arise from features that are specified logically.

Arbitration is the key to interaction resolution — An interaction occurs only when feature requirements are contradictory. Arbitration is the means of automatically weakening the requirements of some features to remove the contradiction.

Choosing Modelling Language — The advantage of a logical approach is self evident: the combination of features is just logical conjunction and the absence of interaction is automatic provided the result is not false. The disadvantage of such an approach is the difficulty in constructing new features and analysing their dynamic behaviour. The principal problem is that of communicating with the clients in such a mathematical model. Contrastingly, more operational (state based) approaches are more powerful with respect to synthesis and analysis. However, it is then much more difficult to say what it means for two features to be contradictory (i.e. have requirements that cannot be met at the same time). The main source of feature interaction problems seems to be exceptions. Using invariants helps to transfer the analysis of exceptions away from the dynamic and

towards a purely static approach. An object oriented approach gives us abstraction and generalisation within a compositional user friendly framework. A combination of a number of semantic frameworks seems to be the only option for all our modelling needs. The probelm of feature interaction is so general, with complex, diverse issues, that we cannot expect a single semantic model approach to be satisfactory.

Operational Requirements are necessary — Concurrent (independent) development of features requires separation of specification from implementation. However, for implementors (designers) to fully understand requirements we expect to be able to animate our specifications. This is also a necessary part of validation. Thus we require operational semantics for our feature specifications (as well as our more abstract logical requirements for testing compatibility).

The incremental development problem: minimise impact of change — In traditional problem domains (and using state-of-the-art development methods) when new functionality is added to a system it is possible to do this by *connecting it* to only a small subset of the system components. (We will not for now attempt to define the different types of connection.) Additions that are localised (with fewer connections) are easier to make than those which are spread about the system. The addition of a feature is inherently a non-local problem (in the current telephone architectures) because it necessitates connection with most of the other components (the other features) in the system. Hence, each addition has global impact. We are searching for an architecture which supports local incrementation techniques.

Restrictive Assumption Approach — Restrict the assumptions that a feature developer can make about the behaviour of its environment (other features in the system included). Since new features will be added later we cannot place any assumptions on them. However, in some cases assumptions must be made. These should be specified as invariant properties that are amenable to static analysis. Our goal is to simplify this analysis by formalising a minimum assumption set that does not restrict our functionality but does eliminate interactions.

3.1.2 Multi-model approaches
— Des approches multi-modèles

SDL[Tur93] is commonly used in the early stages of software development. It provides mechanisms for the specification of data structure, data flow, control flow, encapsulation, information hiding and abstract dependencies, through its support for concurrent objects.

In [GM97] we proposed a mechanism for translating SDL into a TLA$^+$specification [Lam95], in order to provide a proof-theoretical framework. The preservation of properties through the translation is examined within the framework of a simple state-sequence semantics. We identify the strengths and weaknesses of such an approach, and introduce the translation which binds the two different semantics together. We then apply the translation in the verification of two telephone features, and their interaction.

This research arose in response to a need for more formal means of verifying telecom feature systems. We believed that TLA$^+$holds many at-

SDL [Tur93] est communément utilisée dans les premières étapes du développement logiciel. Elle fournit des mécanismes pour la spécification de structures de données, flux de données, flux de contrôle, encapsulation et abstraction, à travers son support pour les objets courants.

Dans [GM97], nous proposions un mécanisme pour traduire SDL vers une spécification TLA$^+$[Lam95], dans le but de fournir un encadrement théorique. La conservation des propriétés à travers la traduction est examinée au sein de l'encadrement d'une simple sémantique *séquence d'états*. Nous identifions les forces et les faiblesses d'une telle approche, et introduisons la traduction qui relie les deux différentes sémantiques ensemble. Nous appliquons alors la traduction dans la vérification des deux *features* téléphoniques, et leur interaction.

Cette recherche est la réponse à un besoin pour davantage de moyens formels de vérification des systèmes de services téléphoniques. Nous croyons que

tractions for rigorous development and so aim to utilise it as our mathematical basis. We also believe that object oriented concepts offer real benefits at all stages of software development. Our strategy is based on combining object oriented and temporal logic models in a coherent and complementary manner. This provides us with a compositional approach to verifying systems of interacting telephone features. In [GM97] we examine the process of generating the first formal design models through a translation from SDL to TLA$^+$.

A number of further case studies, we reinforced our view that no single semantic framework is suitable for the synthesis and analysis of formal feature requirements models, and the choice of modelling language has certain knock-on effects on the transformational design steps which lead to implementation.

Formal languages have a large number of different roles to play. Consequently, choosing a requirements modelling language involves a number of compromises: Should the language be best able to model the abstract or the concrete? Should the language be problem domain specific or problem domain independent? Should the language be oriented towards the client or the engineers? Should the language be informal, rigorous or fully formal? Should the language be state-of-the-art or well-established?

We claim that a multi-language approach is the best way of meeting the needs of both the clients and the engineers. However, the advantages of a single language approach should not be forgotten — conceptual consistency, potential for rigorous design techniques based on correctness preserving transformations, and a common vocabulary between all participants in the development process are all very important. By advocating a mixed model approach we risk losing some, if not all, of these advantages.

la TLA$^+$possède de nombreuses attractions pour un développement rigoureux et vise ainsi à l'utiliser comme notre base mathématique. Nous croyons aussi que les concepts orientés objet offrent de réels bénéfices à toutes les étapes du développement logiciel. Notre stratégie est fondée sur la combinaison de modèles orientés objet et de logique temporelle d'une manière cohérente et complémentaire. Ceci nous fournit une approche compositionelle pour la vérification des systèmes composés. Dans [GM97], nous examinons le procédé pour générer les premiers modèles formels de design à travers une traduction à partir de SDL vers la TLA$^+$.

Dans un certain nombre d'études de cas approfondies, nous renforçons notre position sur le fait qu'aucun encadrement sémantique simple ne convient à la synthèse et à l'analyse de modèles formels des *features*, et le choix de modéliser un langage comporte des implications certaines sur les étapes de la conception transformationelle qui conduisent à l'implémentation.

Les langages formels ont un grand nombre de rôles différents à jouer. Par conséquent, choisir un langage pour la spécification de besoins implique un nombre de compromis : le langage devrait-il au mieux être capable de modéliser l'abstrait ou le concret ? Le langage devrait-il être spécialisé dans le domaine du problème ou indépendant du domaine? Le langage devrait-il être orienté vers le client ou vers les ingénieurs ? Le langage devrait-il être informel, rigoureux ou entièrement formel ? Le langage devrait-il être à la pointe de la technologie ou bien établi ?

Nous affirmons qu'une approche multi-langage est la meilleure façon de répondre à la fois aux besoins des clients mais aussi des ingénieurs. Néanmoins, les avantages de l'approche avec simple language ne devraient pas être oubliés : cohérence conceptuelle, potential pour techniques rigoureuses de la conception basées sur les transformations préservent la sémantique, ainsi que le vocabulaire commun partagé entre tous les participants du processus de développement. Ils sont tous primordiaux. En recommandant une approche modèle mixte, nous risquons de perdre certains de ces avantages, sinon tous.

In [GHM99] we continued to promote a mixed semantic model approach whilst acknowledging that integration is a major concern. The paper introduces a method that incorporates operational state transition models, temporal logic formulae, object oriented structuring mechanisms as well as algebraic formulation. Each of these approaches gives rise to certain advantages and disadvantages, and we advocate a complementary integration which allows the client to express their requirements in the way in which they understand their needs, whilst building formal models for transformation and verification during design.

In our following research, we started our research into compositional verification of safety and liveness/fairness properties in systems of interacting features[GM99a] Without a temporal logic, nondeterminism in the features can be specified only at one level of abstraction: namely that of an internal choice of events. This can lead to many problems in development. For example, consider the specification of a shared database. This database must handle multiple, parallel requests from clients. The order in which these requests are processed is required to be nondeterministic. This is easily specified in a classical object model. However, the requirements are now refined to state that every request must be eventually served (this is a fairness requirement which we cannot directly express in our semantic framework). The only way this can be done is to *over-specify* the requirement by defining how this fairness is to be achieved (for example, by explicitly queueing the requests). This is bad because we are enforcing implementation decisions at the requirements level. With TLA we can express fairness requirements without having to say how these requirements are to be met.

The composition of fairness assumptions in TLA is done at a high level of abstraction and is preserved through the composition process. Fair-

Dans [GHM99], nous continuons de promouvoir une approche modèle mixte demantique (*mixed semantic model*) tout en reconnaissant que l'intégration est un souci majeur. L'essai introduit une méthode qui incorpore "operational state transition models, temporal logic formulae, object oriented structuring mechanisms" ainsi que "algebraic formulation". Chacune de ces approches donne lieu à certains avantages et inconvénients, et nous recommandons une intégration complémentaire qui permet au client d'exprimer ses exigences d'une façon telle qu'ils comprennent leurs besoins, tout en construisant des modèles formels pour une transformation et une vérification pendant la conception.

Dans notre recherche suivante, nous avons débuté celle-ci par la vérification compositionaelle de sécurité et par les propriétés de justice dans des systèmes de *features* [GM99a]. Sans une logique temporelle, le nondéterminisme dans les *features* peut être spécifié seulement à un niveau d'abstraction : à savoir celui d'un choix interne d'événements. Ceci peut conduire à de nombreux problèmes dans le développement. Par exemple, considérons la spécification d'une base de données partagée. Cette base de données doit traiter des requêtes multiples et parallèles de la part des clients. L'ordre dans lequel ces requêtes sont traitées demande à être non-déterministe. Ceci est facilement spécifié dans un modèle d'objet classique. Néanmoins, les besoins sont maintenant affinés pour établir que chaque requête doit être au final servie (ceci est une exigence de justice qui ne peut pas directement s'exprimer dans notre cadre sémantique). La seule manière dont ceci peut être fait est de sur-spécifier le besoin en définissant comment cette justice doit être atteinte (par exemple, en mettant les requêtes dans une file d'attente). Ceci est mauvais parce que nous imposons des décisions d'implémentation au niveau des exigences. Avec la TLA, nous pouvons exprimer des exigences de justice sans avoir à dire comment ces besoins doivent être atteints.

La composition de suppositions de justice dans la TLA est accomplie à un haut niveau d'abstraction et est préservée à travers le processus de composition. Les

ness constraints remove models or traces that do not satisfy them. A service is characterized by a set of flexible variables, initial conditions, a next relation over variables and fairness contraints. When combining two services, we increase the restrictions over traces but we extend the models by adding new variables. TLA provides an abstract way to state fairness assumptions but in our approach this unfriendly syntax is hidden from the customer. We encapsulate fairness within each object as a means of resolving nondeterminism due to internal state transitions. This is a simple yet powerful way for the fairness to be structured and re-used within our requirements models.

This research concluded by stating that the problem of telephone feature interaction is just a particular instance of a general problem in software engineering. The same problem occurs when we consider inheritance in object oriented systems, sharing data in distributed systems, multi-way synchronisation in systems of concurrent processes, etc .... However, the problem is particularly difficult in telephone systems because features are the increments of development.

contraintes de justice enlèvent les modèles ou traces qui ne les satisfont pas. Un service est caractérisé par un ensemble de variables flexibles, de conditions initiales, d'une relation suivante sur des variables et des contraintes de justice. Lorsque deux services sont combinés, nous augmentons les restrictions sur les traces mais nous élargissons les modèles en ajoutant de nouvelles variables. La TLA fournit une façon abstraite pour établir des suppositions de justice mais dans notre approche, cette syntaxe inamicale est cachée du client. Nous cachons la justice au sein de chaque objet comme moyen pour résoudre le non-déterminisme dû à des transitions internes. Ceci est une manière simple mais néanmoins puissante pour que la justice soit structurée et réutilisée au sein de nos modèles de besoins.

Cette recherche se conclut en établissant que le problème de l'interaction des services téléphoniques est juste un simple cas particulier au sein d'un problème général en génie logiciel. Le problème identique apparaît lorsque l'on considère l'héritage dans les systèmes orientés objet, partageant des données dans des systèmes de distribution, une synchronisation dans des systèmes de processus concurrents, etc. Cependant, le problème est particulièrement difficile dans les systèmes de téléphone parce que les *features* sont les incréments du développement.

3.1.3 An algebraic approach: a first step towards a product line — Une approche algébrique : une étape initiale vers la ligne de produit

As we have already seen, the composition (and configuration) of requirements is particularly important in feature specification because the units of incrementation in system development are themselves features. Thus we have requirements models made up of a large number of components, each of which is easy to specify and validate individually, but whose complexity resides in the semantics of composition, and configuration.

In our next research step, we approached the definition of feature composition from the point of view of the client[Gib98]. We identify different ways in which the client would wish to compose their features with the plain old telephone ser-

Comme nous l'avons déjà vu, la composition (et configuration) des besoins est particulièrement importante dans la spécification de *features* parce que les unités d'incrémentation dans le développement de système sont elles-mêmes des *features*. Ainsi, nous avons des modèles de besoins constitués d'un grand nombre de composants, chacun étant aisé à spécifier et à valider individuellement, mais dont la complexité réside dans la sémantique de composition, et de configuration.

Dans notre prochaine étape de recherche, nous faisons une approche de la définition de composition de *features* du point de vue du client [Gib98]. Nous identifions différentes façons dans lesquelles le client souhaiterait composer ses *features* avec POTS. A partir de là,

vice (POTS). From this we motivate the development of a feature composition algebra, the foundation of a feature interaction algebra. Quite simply, we hope to be able to perform a meta-analysis of the feature interaction problem using the feature classes rather than the features themselves. In this paper we showed the type of meta-analysis that can lead to an algebraic formulation of feature composition and configuration.

The most important part of our research was the modelling of an *ideal* environment as a library of features which have already been formally specified and validated. This library will be structured in a class hierarchy where each feature will have many abstract superclasses. The second most important part of our environment is a library of (formally specified) feature composition mechanisms which will operate on a subset of features (depending, of course, on their classification. A re-usable (meta-analysis) will have already identified which classes of features interact and, where possible, will also provide re-usable resolution mechanisms.

Creating a new feature will usually require the client combining already existing features in predefined ways, resulting in a new feature whose classification is calculated automatically following our algebraic semantics. Some features will require the specification of new concepts within the problem domain and, as such, cannot be developed using already existing features. These new features will often *fit directly* into our already existing class hierarchy. In such a case, the meta-analysis does not need to be further extended. In the worse case, new abstract classifications (and composition mechanisms) will be needed and hence the analysis will need to be done *from scratch*. Furthermore, the requirements modellers will be responsible for placing this new case within the formal algebra.

Different types of feature pairs

nous motivons le développement d'un algèbre de composition de *features*. Assez simplement, nous espérons être capable de procéder à une méta-analyse du problème d'interactions, en utilisant les classes de *feature* plutôt que les *features* elles-mêmes. Dans cet essai, nous montrons quel type de méta-analyse peut être conduite vers une formulation algébrique de *feature* composition.

La partie la plus importante de notre recherche a été la modélisation d'un environnement idéal comme une bibliothèque de *features* ayant déjà été formellement spécifiées et validées. Cette bibliothèque sera structurée dans une hiérarchie de classes dans laquelle chaque *feature* aura de nombreuses superclasses abstraites. La deuxième partie la plus importante de notre environnement est une bibliothèque de mécanismes de composition de *features* (formellement spécifiés), qui opèrera sur un sous-ensemble de *features* (ceci dépendant, bien sûr, de leur classification). Une méta-analyse réutilisable aura déjà identifié quelles classes de *features* dialoguent et, là où c'est possible, aura également fourni des mécanismes de résolution réutilisables.

La création d'une nouvelle *feature* nécessite habituellement que le client combine des *features* déjà existantes dans des manières pré-définies, ceci ayant pour résultat une nouvelle *feature* dont la classification est calculée automatiquement suivant notre sémantique algébrique. Certaines features nécessiteront la spécification de nouveaux concepts au sein du domaine du problème et, en tant que telles, ne porront pas être développées en utilisant des *features* déjà existantes. Ces nouvelles *features* s'intègreront souvent directement dans notre hiérarchie de classes pré-existante. Dans un tel cas, la méta-analyse ne nécessite pas d'être davantage extrapolée. Dans le pire des cas, de nouvelles classifications abstraites seront requises et alors l'analyse nécessitera d'être refaite à partir de zéro. Par ailleurs, les modélisateurs de besoins auront la responsabilité de placer ce nouveau cas au sein de l'algèbre formel.

Our case study then examined compositions between pairs of features. We identified 5 different types of feature pairs:

Independent — When the features are combined with POTS, there is no interaction between the features. There is no sharing of actions or state and, thus, no communication between the two features. In other words, the individual behaviour of each feature (with POTS) is exactly as before.

Perfect Friends — The two features do communicate (either through shared state or actions) but this communication in no way alters the behaviour of either feature and so they do not *interact* in the sense used in this report.

Friends — These features do not interact provided some of the nondeterminism in either/both of the models is resolved in some specific way. In some sense, they can be said to make implementation decisions which help their friend feature to work correctly.

Politicians — There is an interaction where both features cannot work exactly as before. However, a resolution mechanism exists whereby some sort of feature priority can be used to resolve the problem automatically. Such a resolution mechanism permits a subset of one or both of the features to be maintained when the two features are combined.

Enemies — There is an interaction and no resolution technique exists other than to say that when one feature is operating the other must be dormant. Two such features can exist in the same system but dynamically both features are never executing at the same time.

The process of formalising such pair-wise analysis was the main goal of our next research.

3.1.4 Compositional Verification of Liveness Properties — Vérification compositionnelle des propriétés de justice

The temporal logic of actions (TLA) provides operators to express liveness requirements in an abstract specification model. TLA does not, however, provide high level composition mechanisms which are essential for synthesising and analysing complex behaviour. Contrastingly, the object oriented paradigm has proven itself in the development of structured specifications. However, most, if not all, of the object oriented formalisms are based on the specification of safety properties and, as such, they do not provide an adequate means of expressing liveness conditions.

We examined [GM99b] how we could combine temporal semantics and object oriented concepts in a complementary fashion. High level re-usable concepts were formalised as different kinds of *fair objects*. The object oriented semantics aid validation and customer communication, whilst the TLA semantics provide a means of formally verifying liveness requirements. The fairness concepts are founded on the notion of objects as servers which may have multiple (concurrent)

La logique temporelle des actions (TLA) permet aux opérateurs d'exprimer des exigences de justice dans un modèle abstrait de spécification. La TLA ne fournit pas, en revanche, des mécanismes de composition de haut niveau, qui sont essentiels pour synthétiser et analyser le comportement complexe. A contrario, le paradigme orienté objet (OO) a fait ses preuves dans le développement de spécifications structurées. Néanmoins, la plus grande partie, sinon l'ensemble, des formalismes OO sont basés sur la spécification des propriétés de sécurité et, en tant que tels, ils ne fournissent pas de moyens adéquats d'expression des conditions de justice.

Nous avons examiné [GM99b] comment nous pouvions combiner sémantique temporelle et des concepts OO d'une façon complémentaire. Des concepts réutilisables de haut niveau ont été formalisés comme différentes sortes de "fair objects". La sémantique OO aide la validation et la communication avec le client, alors que la sémantique TLA apporte un moyen de vérifier formellement les exigences de justice. Les concepts de justice sont fondés sur la notion des objets comme serveurs qui peuvent avoir des clients multi-

clients. Some simple telephone feature specifications illustrated the practical application of our *fair object* semantics.

The goal of this research was to identify high-level fairness concepts within our object oriented framework. The specifications can then be extended to include fairness requirements, which are to be reasoned about using TLA. *Weakly fair objects* and *strongly fair objects* provide us with two such concepts. A major contribution was the formal modelling of five additional high-level concepts, each of which has played an important role in the development of telephone feature specifications.

ples. Quelques simples spécifications de services téléphoniques ont illustré l'application pratique de notre sémantique du "fair objects".

Le but de cette recherche était d'identifier des concepts de justice de haut niveau au sein de notre cadre OO. Les spécifications peuvent alors être étendues pour inclure les exigences de justice, qui doivent être analysées en utilisant la TLA. Les "weakly fair objects" et les "strongly fair objects" nous apportent deux concepts de la sorte. Une contribution majeure a été la modélisation formelle de 5 concepts supplémentaires de haut niveau.

Five additional high-level fairness concepts:

Progression — The notion of progression arises from the way in which concurrent processes are modelled through an interleaving of events. We can "view" such interleaving as though there is a scheduler which randomly chooses which process to be executed at any particular time. In such systems we wish to specify that each of the component processes is *fairly scheduled*.

Possible Fairness — Nondeterminism often gives rise to systems in which it is always possible for an action to be enabled (by following a certain sequence of interal actions) yet the action cannot be guaranteed to be executed through the use of strong fairness or progression. In such cases, we require the notion of *possible fairness*.

Compositional Fairness — We identified many different ways in which we may wish to define new *fair objects* in terms of already specified *fair objects*.

Politeness and Eventuality — We then examined the composition of *fair objects* in such a way that nondeterminism must be resolved by co-operation.

Eventuality Protocols — Finally, we looked at different protocols for sharing responsibility between clients and servers in order to meet such fairness requirements in systems of distributed services.

The next natural progression of this research was to look at compositional verification of systems of fair objects[HGM00]. When following an object-oriented approach to specifying the behaviour of concurrent systems, we would like to be able to specify the liveness properties of each object. However, an object must be viewed as an open system which relies on its environment to ensure that its liveness properties are satisfied. When these objects are composed, the resulting composition must be checked to ensure that the liveness properties specified for each object are preserved. If this is the case, we say that these objects are *po-*

La progression naturelle qui suit dans cette recherche était de regarder la vérification compositionelle des systèmes de "fair objects" [HGM00]. En suivant une approche orientée objet pour spécifier le comportement de systèmes concurrents, nous souhaiterions avoir la possibilité de spécifier les propriétés de justice de chaque objet. Cependant, un objet doit être vu comme un système ouvert, qui dépend de son environnement pour assurer que ses propriétés de justice soient satisfaites. Lorsque ces objets sont composés, la composition résultante doit être vérifiée pour assurer que les propriétés de justice spécifiées pour chaque objet soient préservées. Si c'est bien le cas, nous disons que ces objets sont polis

lite [BFM95, GM99b].

One solution to ensuring that the liveness properties of objects are preserved under composition is to allow only a weak form of object composition in which the liveness properties of the objects are guaranteed to be preserved. This is the approach which is taken in [AL95, CvH99], where only parallel composition of objects (with no communication) is allowed. The liveness properties of the composed object in this case are easy to represent in TLA as the logical conjunction of the liveness properties of each object.

The approach which we take in this paper is to define only *local* liveness requirements on each object. Each object can perform a number of actions, which may be either internal or external. The local liveness requirements for the object specify the liveness of the external actions of the object over all its possible action sequences independent of the environment. We then show how additional liveness constraints can be placed on the internal actions of an object to ensure that it meets its local liveness requirements.

[BFM95, GM99b].

Une solution pour s'assurer que les exigences de justice des objets soient préservées sous composition, c'est de permettre seulement la composition des objets dans laquelle la justice des objets est garantie d'être préservée. Ceci est l'approche qui est prise en [AL95, CvH99], où seulement une composition parallèle des objets (avec aucune communication) est permise. Les propriétés de justice de l'objet composé dans ce cas sont faciles à représenter en TLA comme la conjonction logique des propriétés justice de chaque objet (sous-objet).

L'approche que nous prenons dans cet essai est de définir seulement les exigences locales de justice de chaque objet. Chaque objet peut accomplir un certain nombre d'actions, qui peuvent être ou internes ou externes. Les besoins de jutsice locaux pour l'objet spécifient la justice des actions externes de l'objet, sur toutes ses séquences d'action possibles, indépendantes de l'environnement. Nous montrons alors comment des contraintes supplémentaires de justice peuvent être placées sur des actions internes d'un objet pour s'assurer qu'il remplisse ses exigences de justice locales.

Different levels of politeness between two composed objects

Communication between composed objects is modelled by synchronisation of actions. The composed object therefore has a set of actions consisting of the union of the actions within each individual object (except those actions which are synchronised, which are replaced by a single joint action). Local liveness requirements must also be specified on the external actions for this new composed object. Again, we show how additional liveness constraints can be placed on the internal actions of the composed object to ensure that it meets its local liveness requirements. In the case of synchronised actions, the liveness of the synchronised action is calculated from the liveness of the actions participating in the synchronisation.

Using this method, we can define the following different levels of politeness between the two composed objects:

Independent — if they do not synchronise on any actions

Perfect friends — if they do synchronise on actions, but the internal liveness constraints do not need to be strengthened to meet the local liveness requirements of the composed object

Friends — if they synchronise on actions, but the internal liveness constraints of actions in one or both objects need to be changed to meet the local liveness requirements of the composed object

politicians - if the local liveness requirements of the composed object cannot be met by changing the internal liveness constraints in either object unless some additional resolution mechanism is used

enemies - if the local liveness requirements of the composed object cannot be met by changing the internal liveness constraints in either object or by using any additional resolution mechanism

The main contributions of this paper were therefore to show how liveness requirements on the external actions

of an object can be met by placing liveness constraints on its internal actions, and to show how the liveness requirements of objects composed using a general composition mechanism must be changed to meet the liveness requirements of the composed object.

The final step in this research was to apply our work on composing fair objects to a specific type of common feature interaction[GHM00]. In this work, we formalised the notion of *triggered features* and showed how the resulting model can be used to classify a subset of common interactions. Our underlying *fair object* formal framework, based on the integration of object-state machines and temporal logic, can be exploited to provide support for re-usable analysis. We tested our theoretical results in the construction of object oriented feature requirements models, where each feature is triggered by the same `dialIn` event. Our results support the view that the development of a feature interaction algebra is not just a theoretical proposition but is also a practical engineering possibility.

Many of the classic triggered interaction problems arise when two features are triggered by the same action and the resulting introduction of nondeterminism leads to inconsisent requirements. The most common technique to resolve such interactions is to use a priority mechanism. We formulated a refinement of a system with two interacting triggered features which resolves the interaction automatically. The idea is that the first time the non-determinism arises then the user is forced to make the choice. Consequently, any time after this choice has been made, the nondeterminism will be resolved in a manner consistent with the user's initial choice. This type of refinement could be made available as a correctness preserving transformation for use during design: instead of the designer having to resolve the interaction statically, they can gurantee that the resolution will be carried out dynamically. In this case, the designer's goal of having consistent resolution would have automated tool support.

L'étape finale dans cette recherche était d'appliquer notre travail à la composition de "fair objects" pour un type spécifique d'interaction de service commune [GHM00]. Dans ce travail, nous formalisons la notion de "triggered features" et montrons comment le modèle résultant peut être utilisé pour classifier un sous-ensemble d'interactions communes. Notre cadre formel de "fair object", basé sur l'intégration de "object state machines" et de logique temporelle, peut être exploité pour apporter un support pour une analyse réutilisable. Nous avons testé nos résultats théoriques dans la construction de modèles OO de *features*, où chaque *feature* est déclenchée par le même `dialIn` evenement. Nos résultats supportent l'avis que le développement d'un "feature interaction algebra" n'est pas simplement une proposition théorique mais aussi une possibilité d'ingénierie pratique.

Nombre des problèmes classiques de " triggered interaction" surgissent lorsque deux *features* sont déclenchées par la même action ; et l'introduction résultante de non-déterminisme conduit à des exigences inégales. La technique la plus commune pour résoudre de telles interactions est d'utiliser un mécanisme de priorité. Nous avons formulé un raffinement d'un système avec deux "interacting triggered features", qui résout automatiquement l'interaction. L'idée est que la première fois que le non-déterminisme survient, alors l'utilisateur est forcé de faire un choix. Par conséquent, à tout moment après que ce choix est fait, le non-déterminisme sera résolu d'une manière en cohérence avec le choix initial de l'utilisateur. Ce type de raffinement pourrait être disponible en tant que *correctness preserving transformation* pour l'utiliser lors de la conception : au lieu que les developeurs ne doivent résoudre l'interaction de façon statique, ils peuvent garantir que la résolution sera conduite de façon dynamique. Dans ce cas, le but des designers d'avoir une résolution cohérente aura un support outil automatisé.

The concept of stability may lead to an alternative solution where the invariants of each feature are weakened so that they have only to be true when that feature has actually been triggered — provided that they will always eventually be true then we do not mind if they are false while another feature is executing. We proposed the need for new synchronisation mechanisms which incorporate stability into their semantics in order to facilitate the exiting of unstable states by prioritising those states which are stable (when an internal choice of actions is provided then a system which is constructed from the stable synchronisation operators will choose to *move towards a stable state*.) This remains a very challenging research area: the designer's goal of automatically keeping the system as stable as possible would require some sort of self managing behaviour.

Le concept de stabilité peut conduire à une solution alternative où les invariants de chaque *feature* sont affaiblis afin de devoir être vrais quand cette *feature* a, en fait, été déclenchée — à condition qu'ils soient toujours vrais au final alors cela ne nous gène pas s'ils sont faux pendant qu'une autre *feature* exécute. Nous avons suggéré que de nouveaux mécanismes de synchronisation incorporent une stabilité dans leur sémantique soient necessaires ; ceci — dans le but de faciliter l'évacuation des états instables en donnant la priorité aux états qui sont stables (quand un choix interne d'actions est fourni alors un système, construit à partir des opérateurs stables de synchronisation, choisira de se déplacer un état stable). Ceci est un domaine de recherche comportant un grand challenge : l'objectif du développeur de garder automatiquement le système aussi stable que possible exigerait une sorte de comportement d'autogestion.

3.1.5 Important Technical Contribution: Fair Object Composition

There is currently much research in model driven development, service oriented architectures and automated composition of services. A major outstanding issue is being able to compose in a safe manner: so that new compositions do not inadvertently break (safety and liveness) requirements that were previously being met. The research on fair object composition, that is reported earlier in this section, is an important technical contribution towards a possible solution to the problem of scaleable verification techniques, in the development of interacting systems of services that are required to be "fair". In this subsection we provide some additional technical details (for the interested reader). The original paper — *Composing Fair Objects*[GHM00] — provides a more complete report.

As an example of a fair object, consider a double-ended queue as shown in figure 1.

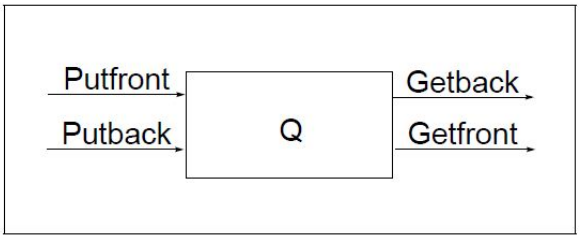

Figure 1: A double ended queue

The queue object provides operations to put or get elements to the front or back of the queue (these are all external actions). This could be defined as follows in TLA:

$$
\begin{aligned}
Addfront(n) &\triangleq \land\ Len(contents) < 100 \\
&\quad \land\ contents' =< n > \circ contents \\
\\
Addback(n) &\triangleq \land\ Len(contents) < 100 \\
&\quad \land\ contents' = contents \circ < n > \\
\\
Getfront(n) &\triangleq \land\ Len(contents) > 0 \\
&\quad \land\ contents =< n > \circ contents' \\
\\
Getback(n) &\triangleq \land\ Len(contents) > 0 \\
&\quad \land\ contents = contents' \circ < n > \\
\\
Step &\triangleq \exists n \in Nat : \lor\ Addfront(n) \\
&\qquad\qquad\ \lor\ Addback(n) \\
&\qquad\qquad\ \lor\ Getfront(n) \\
&\qquad\qquad\ \lor\ Getback(n) \\
\\
Init &\triangleq contents =<> \\
\\
Spec &\triangleq \exists contents \in Seq(Nat) : Init \land \Box[Step]_{contents}
\end{aligned}
$$

Figure 2: A double ended queue in TLA

The state variable $contents$ is a sequence which holds the elements of the queue (up to a maximum of 100 elements). We would like to guarantee that the actions $Getfront$ and $Getback$ will always eventually be enabled, but this relies on at least one of the external actions $Putfront$ and $Putback$ always eventually taking place. Similarly, we would like to guarantee that the actions $Putfront$ and $Putback$ will always eventually be enabled, but this relies on at least one of the external actions $Getfront$ and $Getback$ always eventually taking place. These liveness requirements can be defined in an assumption/guarantee style as follows:

$$
\begin{aligned}
&\exists n \in Nat : \\
&\Box\Diamond(Putfront(n) \lor Putback(n)) \land \\
&\Box\Diamond(Getfront(n) \lor Getback(n)) \Rightarrow \\
&\Box\Diamond(Enabled\, Getfront(n) \land Enabled\, Getback(n)) \land \\
&\Box\Diamond(Enabled\, Putfront(n) \land Enabled\, Putback(n))
\end{aligned}
$$

Figure 3: Liveness Requirement in TLA

There are a number of different object composition mechanisms we could use. One such mechanism is purely parallel composition, in which there is no communication between the composed objects. The liveness properties of the composed object in this case are represented as the logical conjunction of the liveness properties of each individual object. Fairly simple proof obligations must be discharged to show that the liveness assumptions of the composed objects are preserved.

In general, however, we require stronger forms of object composition in which there is communication between the composed objects, which means that the liveness properties of the objects may be affected. We specify object

composition by the synchronisation of actions as follows:

$$O_1 || [< \mathcal{A}_1^1, \mathcal{A}_1^2 >, \ldots, < \mathcal{A}_n^1, \mathcal{A}_n^2 >] || O_2$$

where each external action $\mathcal{A}_i^1$ in the object O_1 synchronises with the corresponding external action $\mathcal{A}_i^2$ in the object O_2. Consider the following two fair objects:

$O_1 \triangleq \exists x_1 : Init_1 \wedge \Box[Ext_1 \vee Int_1]_{x_1} \wedge F_1$
$O_2 \triangleq \exists x_2 : Init_2 \wedge \Box[Ext_2 \vee Int_2]_{x_2} \wedge F_2$

The result of composing these two objects is as follows:

$O_1 || [< \mathcal{A}_1^1, \mathcal{A}_1^2 >, \ldots, < \mathcal{A}_n^1, \mathcal{A}_n^2 >] || O_2 \triangleq$
$\exists x_1, x_2 : Init_{12} \wedge \Box[Ext_{12} \vee Int_{12}]_{<x_1, x_2>} \wedge F_{12}$
where
$Init_{12} \triangleq Init_1 \wedge Init_2$
$Ext_1 \triangleq \mathcal{A}_1^1 \vee \ldots \vee \mathcal{A}_n^1 \vee \ldots \vee \mathcal{A}_{n+k_1}^1$
$Ext_2 \triangleq \mathcal{A}_1^2 \vee \ldots \vee \mathcal{A}_n^2 \vee \ldots \vee \mathcal{A}_{n+k_2}^2$
$Ext_{12} \triangleq \mathcal{A}_{n+1}^1 \vee \ldots \vee \mathcal{A}_{n+k_1}^1 \vee \mathcal{A}_{n+1}^2 \vee \ldots \vee \mathcal{A}_{n+k_2}^2$
$Int_{12} \triangleq Int_1 \vee Int_2 \vee \mathcal{A}_1^{12} \vee \ldots \vee \mathcal{A}_n^{12}$
$\mathcal{A}_i^{12} \triangleq \mathcal{A}_i^1 \wedge \mathcal{A}_1^2$
$F_{12} \triangleq F_1 \wedge F_2$

The external actions in the resulting composition are those external actions in the two objects which are not involved in synchronisations; synchronised actions become internal. The initial state of the resulting composition is the conjunction of the initial states of each object. Similarly, the fairness constraints on the resulting composition is the conjunction of the fairness constraints on each object. The synchronised actions are also the conjunction of the individual actions involved in the synchronisation. For example, consider the composition of two double-ended queues to produce another double-ended queue as shown in figure 4.

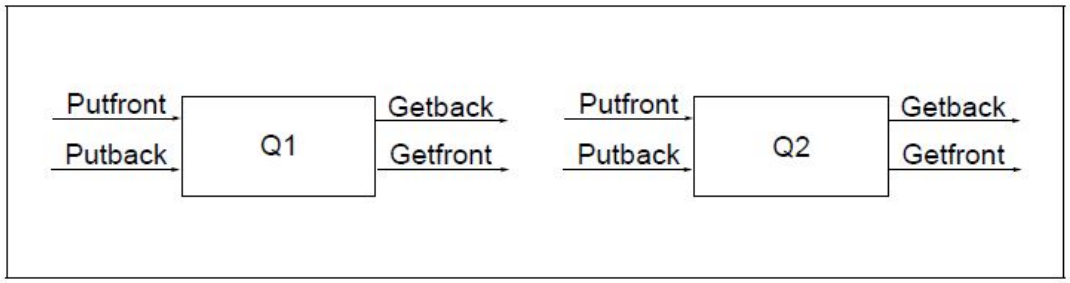

Figure 4: Composed double-ended queues

This composition, and the TLA specification resulting from this specification are as follows:

$$Q_1 |[<Getback, Putfront>, <Getfront, Putback>]| Q_2$$

$$Addfront(n) \triangleq \begin{array}{l} \wedge\, Len(Q1.contents) < 100 \\ \wedge\, Q1.contents' =< n > \circ\, Q1.contents \end{array}$$

$$Addback(n) \triangleq \begin{array}{l} \wedge\, Len(Q2.contents) < 100 \\ \wedge\, Q2.contents' = Q2.contents \circ < n > \end{array}$$

$$Getfront(n) \triangleq \begin{array}{l} \wedge\, Len(Q1.contents) > 0 \\ \wedge\, Q1.contents =< n > \circ\, Q1.contents' \end{array}$$

$$Getback(n) \triangleq \begin{array}{l} \wedge\, Len(Q2.contents) > 0 \\ \wedge\, Q2.contents = Q2.contents' \circ < n > \end{array}$$

$$Internal1 \triangleq \begin{array}{l} \wedge\, Len(Q1.contents) > 0 \\ \wedge\, Q1.contents = Q1.contents' \circ < n > \\ \wedge\, Len(Q2.contents) < 100 \\ \wedge\, Q2.contents' =< n > \circ\, Q2.contents \end{array}$$

$$Internal2 \triangleq \begin{array}{l} \wedge\, Len(Q2.contents) > 0 \\ \wedge\, Q2.contents =< n > \circ\, Q2.contents' \\ \wedge\, Len(Q1.contents) < 100 \\ \wedge\, Q1.contents' = Q1.contents \circ < n > \end{array}$$

$$Step \triangleq \exists n \in Nat : \begin{array}{l} \vee\, Addfront(n) \\ \vee\, Addback(n) \\ \vee\, Getfront(n) \\ \vee\, Getback(n) \\ \vee\, Internal1 \\ \vee\, Internal2 \end{array}$$

$$Init \triangleq Q1.contents =<> \wedge\, Q2.contents =<>$$

$$Spec \triangleq \exists Q1.contents, Q2.contents \in Seq(Nat) : Init \wedge \Box[Step]_{<Q1.contents, Q2.contents>}$$

This composed object will operate normally as a double-ended queue, provided that the environmental liveness requirements for each of the composed objects are upheld. However, it is possible that the actions on which the two objects synchronise will cause their environmental liveness requirements to be broken.

For example, consider the synchronisation between $Q1.Getback$ and $Q2.Putfront$. If the environmental liveness requirements for both these objects were satisfied, then both of these actions would be always eventually enabled, but it is possible that they will never actually be enabled at the same time. The internal action $Internal1$ may therefore never actually be taken. Similarly, the internal action $Internal2$ may never be taken. The environmental liveness requirements for each object have therefore been broken.

Liveness requirements which are broken due to synchronisation may be resolved by placing additional fairness constraints on the synchronised actions. In order to determine the fairness constraint which must placed on synchronised actions, we need to determine their level of liveness. A synchronised action is enabled only if the individual actions involved in the synchronisation are enabled at the same time within each object. We can therefore determine the liveness of a synchronised action from the liveness of the individual actions involved in the synchronisation. The four different types of liveness that we consider are:

- EA (Eventually always),
- AE(Always Eventually),
- APE (Always Possibly Eventually), and
- PAE (Possibly Always Eventually)

We might expect that the liveness of a synchronised action is given by some least upper bound of the liveness of the individual actions involved in the synchronisation. However, the example of the double-ended queues shows that this is not necessarily the case. Even if the actions involved in the synchronisation are always eventually

enabled, they may never actually be enabled at the same time. The synchronised action in this case will only be always possibly eventually enabled.

The liveness of a synchronised action can be determined from the liveness of the individual actions involved in the synchronisation according to the following table:

	Object 1			
Object 2	EA	AE	APE	PAE
EA	EA	AE	APE	PAE
AE	AE	APE	APE	PAE
APE	APE	APE	APE	PAE
PAE	PAE	PAE	PAE	PAE

Each of the entries in this table have been proved on an individual basis. (The details of these proofs are not given here.) Now that the liveness of synchronised actions can be determined, the fairness constraints which must be placed on them to satisfy liveness requirements can be determined. For example, in the case of the composed double-ended queues we can determine that the synchronised actions $Internal1$ and $Internal2$ are always possibly eventually enabled. Hyperfairness constraints must therefore be placed on these actions to ensure that the liveness requirements of the composed objects are satisfied.

Thus, our verification of fairness in a system of interacting objects is compositional. Potential state explosion problems are avoided, and correctness can be guaranteed through the application of predefined composition mechanisms. However, this approach is limited by restricting the types of composition that can be used in building the system. This trade-off — between richness of composition and simplicity of verification — is fundamental to all formal software engineering.

Feature Composition Bibliography

[AL95] M. Abadi and L. Lamport. Conjoining Specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–533, May 1995.

[BFM95] N. Barreiro, J.L. Fiadeiro, and T. Maibaum. Politeness in Object Societies. In R. Wieringa and R. Feenstra, editors, *Information Systems: Correctness and Reusability*, pages 119–134. World Scientific Publishing Company, 1995.

[CvH99] E. Canver and F.W. von Henke. Formal development of object-based systems in a temporal logic setting. In *Formal Methods for Open Object-Based Distributed Systems*, pages 419–436. Kluwer Academic Publishers, February 1999.

[GHM99] J. Paul Gibson, Geoff Hamilton, and Dominique Méry. Integration problems in telephone feature requirements. In Keijiro Araki, Andy Galloway, and Kenji Taguchi, editors, *Integrated Formal Methods, Proceedings of the 1st International Conference on Integrated Formal Methods (IFM 99)*, pages 129–148, York, UK, June 1999. Springer.

[GHM00] J. Paul Gibson, Geoff Hamilton, and Dominique Méry. A taxonomy for triggered interactions using fair object semantics. In Muffy Calder and Evan H. Magill, editors, *Feature Interactions in Telecom-*

munications and Software Systems VI (FIW 2000), pages 193–209, Glasgow, Scotland, UK, 2000. IOS Press.

[Gib97] J. Paul Gibson. Feature requirements models: Understanding interactions. In Petre Dini, Raouf Boutaba, and Luigi Logrippo, editors, *Feature Interactions in Telecommunications Networks IV, (FIW 1997)*, pages 46–60, Montréal, Canada, 1997. IOS Press.

[Gib98] J. Paul Gibson. Towards a feature interaction algebra. In Kristofer Kimbler and Wiet Bouma, editors, *Feature Interactions in Telecommunications and Software Systems V (FIW 1998)*, pages 217–231, Malmö, Sweden, 1998. IOS Press.

[GLR11] J. Paul Gibson, Eric Lallet, and Jean-Luc Raffy. Formal object oriented development of a voting system test oracle. *Innovations in Systems and Software Engineering (Special issue UML-FM11)*, 2011. To appear.

[GM97] J. Paul Gibson and Dominique Méry. Telephone feature verification: Translating SDL to TLA+. In Ana R. Cavalli and Amardeo Sarma, editors, *SDL '97 Time for Testing, SDL, MSC and Trends — 8th International SDL Forum*, pages 103–118, Evry, France, September 1997. Elsevier.

[GM99a] J. Paul Gibson and Dominique Méry. Formal modelling of services for getting a better understanding of the feature interaction problem. In Dines Bjørner, Manfred Broy, and Alexandre V. Zamulin, editors, *PSI '99: Proceedings of the Third International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, volume 1755 of *Lecture Notes in Computer Science*, pages 155–179, Akademgorodok, Novosibirsk, Russia, 1999. Springer.

[GM99b] Paul Gibson and Dominique Méry. Fair objects. *Object-oriented technology and computing systems re-engineering*, pages 122–140, 1999.

[HGM00] Geoff Hamilton, J. Paul Gibson, and Dominique Méry. Composing fair objects. In Fouchal and Lee, editors, *International Conference on Software Engineering Applied to Networking and Parallel/Distributed Computing (SNPD '00)*, pages 225–233, Reims, France, May 2000.

[Lam95] L. Lamport. TLA$^+$. Technical report, December, 5th july 1995.

[Tur93] K.J.T. Turner. *Using FDTS: An Introduction To ESTELLE, LOTOS and SDL*. John Wiley and Sons, 1993.

[Zav93] Pamela Zave. Feature interactions and formal specifications in telecommunications. *IEEE Computer*, 26(8):20–30, 1993.

3.2 Software Engineering Applied Theory: E-voting Systems — Théorie appliquée du génie logiciel : les systèmes de vote électronique

3.2.1 Electronic Voting is Safety Critical — Le vote électronique doit être consideré comme une système critique

In 2003, we first began to consider e-voting as an example of a critical system that should be

En 2003, nous avons tout d'abord commencé à considérer le vote électronique comme exemple de système

developed using formal methods[MG03]. At this time, the Irish government had already begun to introduce an electronic voting system, stating that it would be easier to use, give more accurate results, eliminate spoiled votes, speed up the count, and modernise the electoral system.

The wider computer science community had not been consulted in the choice or development of this system, and it had become apparent that there were several reasons to be concerned about the system chosen.

The aims of this report were to present a list of criteria that any electronic voting system must meet in order to make it an acceptable replacement for a paper system, and then to demonstrate that the system currently being introduced by the Irish government did not, in fact, meet those criteria.

critique qui devrait être développé en utilisant les méthodes formelles [MG03]. A ce moment-là, le gouvernement irlandais avait déjà commencé à introduire un système de vote électronique, avançant que ce serait plus facile à utiliser, que cela donnerait des résultats plus précis, que cela éliminerait les votes nuls, que cela rendrait le dépouillement plus rapide, et que cela moderniserait le système électoral.

La communauté élargie des sciences informatiques n'avait pas été consultée dans le choix ni dans le développement de ce système, et il est apparu évident qu'il existait plusieurs raisons de s'inquiéter concernant le système choisi.

L'objectif de ce rapport était de présenter une liste de critères que tout système de vote électronique doit respecter afin de le rendre acceptable en tant que remplacement du système papier, et ainsi démontrer que le système étant actuellement introduit par le gouvernement irlandais ne respectait pas, en fait, ces critères.

3.2.2 Formalising the e-voting count algorithms — Formalisant les algorithmes de dépouillement du vote électronique

From a software engineering view point, it was difficult to understand how errors could have been made in implementing the count algorithms in exisiting systems. In order to examine this problem, we first asked whether formal specifications of these algorithms would be easier to validate than informal descriptions[CEB+05]. This paper presented a pilot study which compared the comprehensibility of two specifications: a formal specification and an informal specification. The two documents used in the pilot study implemented the same logic, namely a portion of the Irish Electoral system. The "informal specification" was taken directly from the legal definition of the count rules for Irish elections.

The results of this initial study were inconclusive; but did lead to the development of the specification using an algebraic specification language[Gib05].

D'un point de vue du génie logiciel, il était difficile de comprendre comment les erreurs avaient pu être faites en codant les algorithmes de comptage dans les systèmes existants. Dans le but d'examiner ce problème, nous nous sommes, en premier lieu, interrogés à savoir si les spécifications formelles de ces algorithmes seraient plus faciles à valider que des descriptions informelles [CEB+05]. Cet essai présentait une étude pilote qui comparait la compréhensibilité des deux spécifications : une spécification formelle et une spécification informelle. Les deux documents utilisés dans l'étude pilote implémentaient la même logique, à savoir une portion du système électoral irlandais. La "spécification informelle" était tirée directement de la définition légale des règles électorales pour les élections irlandaises.

Les résultats de l'étude initiale étaient peu concluants, mais ils ont tout de même conduit au développement d'une spécification utilisant un langage algébrique de spécification [Gib05].

3.2.3 E-voting requirements engineering — L'ingénierie des besoins du vote électronique

From our analysis of the failure of e-voting systems around the world, it was clear that poorly specified requirements were a major problem. Subsequently, we examined the question of whether this was a result of manufacturers not meeting inetrnational standards, or whether the blame could be applied to the poor quality of the standards themselves[MG06]. We focused on the Council of Europe (CoE) recommendations and instruments.

The list of 12 instruments analysed covered a diverse range of documents, including the *Universal Declaration on Human Rights*, the *European Charter of Local Self-Government* and the *Convention on Cybercrime*. It also included the *Code of Good Practice in Electoral Matters*, which was produced by the Venice Commission (VC).

We demonstrated that this inter-related set of complex documents is analagous to a software system which has evolved over time, in response to ever changing sets of requirements. The system depends on a large number of other systems, and the environment of the system (the context in which it is being used) is not clearly understood. With such legacy systems, one often reaches a stage where the system's operation can only be maintained through a re-structuring (re-engineering) of the system and its architecture. Many techniques exist for this task, one of which is known as reverse engineering. We proposed reverse engineering of the e-voting standards, with focus on arriving at a set of documents that can be usefully applied at the requirements capture stage of e-voting development.

We demonstrated how the standards would benefit from being re-engineered, and we analysed whether they — as they are stated — adhere to good practice with respect to system analysis and requirements engineering. Our goal was not

A partir de notre analyse des échecs de systèmes électroniques à travers le monde, il était clair que des exigences pauvrement spécifiées représentaient le problème majeur. Par la suite, nous avons examiné la question de savoir si c'était le résultat de fabricants ne répondant pas aux standards internationaux, ou si les reproches pouvaient être appliquées à la pauvre qualité des seuls standards [MG06]. Nous nous sommes focalisés sur les recommandations et instruments relatifs au Conseil de l'Europe (CoE).

La liste des 12 instruments analysés couvrait un éventail divers de documents, incluant *La Déclaration Universelle des Droits de l'Homme*, *La Charte Européenne de l'Autonomie Locale* et *La Convention sur la Cybercriminalité*. Etait également inclus *Le Code de Bonne Conduite en Matière Electorale*, qui a été produit par La Commission de Venise (VC).

Nous avons démontré que cet ensemble interdépendant de documents complexes est analogue à un système de logiciel qui a évolué à travers le temps, en réponse à des ensembles de besoin en constant changement. Le système dépend d'un grand nombre d'autres systèmes, et l'environnement du système (le contexte dans lequel il a été utilisé) n'est pas clairement compris. Avec de tels systèmes, on atteint souvent une étape où l'opération du système ne peut être maintenue qu'à travers une restructuration du système et de son architecture. De nombreuses techniques existent pour cette tâche, l'une d'elles est connue comme le "reverse engineering". Nous avons proposé un "reverse engineering" des standards du vote électronique, en accentuant sur le fait d'arriver à un ensemble de documents qui peuvent être appliqués de façon utile à l'étape de l'ingénierie des besoins du développement du vote électronique.

Nous avons démontré comment les standards bénéficieraient d'une reconstruction, et nous avons analysé — comme ils sont établis — s'ils adhèrent à la bonne pratique en ce qui concerne l'analyse de système et l'ingénierie des besoins. Notre objectif était de ne pas

to say whether we agree or disagree with the standards. Our aim was to show that the way in which the standards are expressed is very poor, in the sense that it makes it almost impossible for them to achieve both: their objectives, as defined by the VC; and fundamental software engineering quality criteria.

Following on from this work, we examined the problem of maintaining e-voting standards[GM08]. It is clear that e-voting systems should be verified to be fit-for-purpose before being deployed, but that there is a serious lack of provision for verification and maintenance in existing standards and recommendations for e-voting. A change to requirements, or to the system, usually results in the previously established fitness-for-purpose being compromised. Therefore change must be managed, and standards documents must make provision for their own maintenance. Verification is a process of establishing a relationship between what is required of the system and properties of the actual system. It is good practice that an independent authority be responsible for verification of systems against requirements. It must be possible to determine whether a given authority can be trusted to fulfil this task competently. Thus, requirements documents must not only say what standards are to be met, but must also state the minimum capabilities expected of any testing authority.

The whole e-voting system development process is prone to human-error. This applies to the requirements, standards and the systems they describe. We must introduce suitable procedures for dealing with these errors, including the identification of responsible parties. We must also ensure that there is adequate incentive for the correction of errors. If maintenance of systems requires expensive recertification, there is a risk that vendors will not make necessary changes to their systems (to avoid recertification) or will make changes

dire si nous étions en accord ou en désaccord avec les standards. Notre but était de montrer que la manière dont les standards sont exprimés est très pauvre, dans le sens où cela devient pour eux pratiquement impossible d'atteindre à la fois : les objectifs, ainsi définis par la VC, et les critères de qualité fondamentaux en génie logiciel.

A la suite de ce travail, nous avons examiné le problème du maintien des standards du vote électronique [GM08]. Il est clair que les systèmes de vote électronique devraient être vérifiés pour être *"fit-for-purpose"* avant d'être déployés ; mais il est clair aussi qu'il y a un sérieux manque de dispositions pour la vérification et la maintenance des standards existants et des recommandations pour le vote électronique. Un changement aux besoins, ou au système, résulte habituellement dans la compromission du *fitness-for-purpose* précédemment établi. Donc le changement doit être géré, et les documents de standards doivent faire un approvisionnement pour leur propre maintenance. La vérification est un processus d'établissement d'une relation entre ce qui est requis pour le système et les propriétés du système réel. Il est de bonne pratique qu'une autorité indépendante soit responsable pour la vérification des systèmes contre les besoins. Il doit être possible de déterminer si on peut faire confiance à une autorité donnée pour remplir cette tâche de façon compétente. Ainsi, les documents de besoins ne doivent pas seulement stipuler que des standards doivent être respectés, mais doivent aussi établir les aptitudes minimumes attendues de la part de n'importe quelle autorité faisant le test.

Tout le processus de développement du système de vote électronique est enclin aux erreurs humaines. Ceci s'applique aux besoins, aux standards et aux systèmes qu'ils décrivent. Nous devons introduire des procédures adéquates pour faire face à ces erreurs, en incluant l'identification des parties responsables. Nous devons également nous assurer qu'il existe une motivation pour corriger ces erreurs. Si la maintenance des systèmes requiert une re-certification onéreuse, il y a risque que les vendeurs ne fassent pas les changements nécessaires à

without having the systems recertified.

Error discovery is not the only agent of change for requirements and systems. For example, the introduction of new legislation, new election types, or new technology will have direct consequences. This requires careful co-ordination between all concerned parties. Whenever a system changes, whatever the surrounding circumstances, it must be tested and re-certified. However, if the system under evaluation has been well-engineered, it may not be necessary to begin again with every modification. In this paper we examined what it means for a system to be well-engineered and proposed maintenance procedures specific to the problem of e-voting.

In 2009, there was a large community of researchers working on e-voting requirements engineering. Thus, it seemed natural to organise a research workshop on the topic[GJ09]. The workshop examined best-practice for the engineering of e-voting system requirements, and the specification of e-voting system standards and accreditation (certification) processes. Papers compared and contrasted requirements engineering processes and documents which predate electronic voting systems with those that motivate the development of future voting systems. The workshop highlighted the importance of case studies — with some papers addressing evaluation of existing commercial systems and others analysing prototypes that have developed out of research environments.

3.2.4 E-voting and formal methods
Vote électronique et méthodes formelles

3.2.4.1 Verifying the interface
— Vérification de l'interface

In [CGM07b] we demonstrate the use of the formal method B in guaranteeing simple safety properties of a voting machine implementing a common variation of the single transferable vote

leurs systèmes (pour éviter la re-certification) ou fassent des changements sans avoir re-certifié les systèmes.

La découverte d'erreurs n'est pas le seul agent de change pour les besoins et les systèmes. Par exemple, l'introduction d'une nouvelle législation, de nouveaux types d'élection, ou d'une nouvelle technologie aura des conséquences directes. Ceci exige une coordination prudente entre les parties concernées. Lorsqu'un système change — quelles qu'en soient les circonstances l'entourant — il doit être testé et re-certifié. Cependant, si le système sous évaluation a été bien construit, il ne sera peut-être pas nécessaire de recommencer avec chaque modification. Dans cet essai, nous examinons ce que cela signifie pour un système d'être bien construit et nous avons proposé des procédures spécifiques de maintenance au problème du vote électronique.

En 2009, une grande communauté de chercheurs travaillait sur l'ingénierie de besoins du vote électronique. Aussi, il est apparu naturel d'organiser des ateliers sur le thème [GJ09]. L'atelier a examiné *best-practice* pour l'ingénierie des besoins du système de vote électronique et la spécification des standards de système de vote électronique et les processus d'accréditation (certification). Des essais ont comparé et contrasté des procédés et documents qui sont antérieurs aux systèmes de vote électronique avec ceux qui motivent le développement de futurs systèmes de vote. L'atelier a mis en avant l'importance d'études de cas — avec des essais traitant de l'évaluation de systèmes commerciaux existants, et d'autres analysant des prototypes qui se sont développés à partir d'environnements de recherche.

Dans [CGM07b], nous démontrons l'utilisation de la méthode formelle B en garantissant des propriétés de sécurité simples pour une machine de vote implémentant une variation habituelle du "single transferable

(STV) election process. The properties we examined are concerned with the collection and storage of only *valid* votes. We demonstrate that guaranteeing *validity* not only helps in the formal verification of the counting process, but also has an important role to play in making the machine more secure. Using the B-method, we applied an incremental refinement approach to verifying a sequence of designs for the collection and storage of votes, which we prove to be correct with respect to the simple requirement.

vote"(STV[1]) election process. Les propriétés que nous avons examinées sont concernées par la collection et le stockage seulement des votes valables. Nous démontrons que garantir la validité aide non seulement dans la vérification formelle du processus de comptage, mais a également un rôle important à jouer en rendant la machine plus sûre. En utilisant la méthode B, nous appliquons une approche incrémentielle de raffinement à la vérification d'une séquence de conceptions pour la collection et le stockage des votes, ce que nous prouvons comme étant correct par rapport à la simple exigence.

1 Vote unique et transférable.

Clearly, if invalid votes manage to get passed to the tabulation process (to be counted) then there is a risk that this could break the counting process. For example, it would not be unreasonable to suggest that some of the tabulation methods make the assumption that the votes being counted are valid. However, without some degree of formal verification it is also likely that an invalid vote could — by accident — be counted and that this could lead to an incorrect result, a run-time error, or non-termination. Consequently, this weakness could also be exploited by an attacker to deliberately manipulate the election process.

Such a potential attack is similar to those mentioned in [MKSW06] where the security of votes stored in memory is addressed. In particular, the use of Trojan code to exploit vote data that has been tampered with is shown to be a real threat that requires elaborate schemes for the secure storage of votes. In most e-voting systems, there is a clear interface between the the storage of votes and the input of votes. We argue that the same degree of care must be taken in designing the vote interface to ensure that Trojan code cannot be used to exploit the input of invalid votes and their subsequent transfer to long term storage.

3.2.4.2 Verifying storage
— Vérification du stockage

In [CGM07a] we illustrate the utility of our refinement-based approach by verifying — through the application of a reusable formal design pattern — a store design that uses a specific PROM technology and applies a specific encoding mechanism. The main property that we examined was concerned with the need for *tamper-evident* storage, which addresses the risk of unauthorised tampering of vote data after it has been correctly registered and stored.

Dans [CGM07a], nous illustrons l'utilité de notre approche basée sur la raffinement en vérifiant — à travers l'application d'un patron formel de conception réutilisable — une conception de stockage qui utilise une technologie PROM spécifique et qui applique un mécanisme spécifique d'encodage. La propriété principale que nous avons examinée était concernée par le besoin d'un stockage *"tamper-evident"*, qui traite du risque de falsification non autorisé des données électorales, après avoir été correctement enregistrées et stockées.

The storage of votes is a critical component of any voting system. In traditional systems there is a high level of transparency in the mechanisms used to store votes, and thus a reasonable degree of trustworthiness in the security

of the votes in storage. This degree of transparency is much more difficult to attain in electronic voting systems, and so the specific mechanisms put in place to ensure the security of stored votes require much stronger verification in order for them to be trusted by the public. There are many desirable properties that one could reasonably expect a vote store to exhibit. From the point of view of security, we argue that *tamper-evident* storage is one of the most important requirements: the changing, or deletion of already validated and stored votes should be detectable; as should the addition of unauthorised votes after the election is concluded. We propose the application of formal methods (in this paper, event-B) for guaranteeing, through construction, the correctness of a vote store with respect to the requirement for *tamper-evident* storage.

In *Analysis of an electronic voting system*[KSRW04], we see that such a security weakness already exists in one of the most widely procured voting systems:

> "...an adversary could alter election results by modifying ballot definition files, and ...it leaves no evidence that an attack was ever mounted"

Here, the "adversary" is most likely to be a single insider (election official) with access to the storage device. We argue that it is the responsibility of the storage designers to guarantee the security of the votes stored without having to make an assumption about the behaviour or intent of such officials.

In order to illustrate how a guarantee could be made, we used event-B and applied an incremental refinement approach to verifying a sequence of designs for the storage of votes, which we proved to be correct-through-construction with respect to the simple requirement that the vote storage is tamper-evident.

The main design that is modelled and verified in this paper is taken directly from the work by Molnar, Kohno, Sastry and Wagner[MKSW06]. Their proposed solution to providing tamper-evident storage involves the application of Manchester codes[Sta85] and a write-once data PROM store. The encoding simply represents a 0 as a 01 and a 1 as a 10. Thus, when validating votes stored as pairs of bits there are 2 additional pair cases to be considered, where (because our memory allows only 1s to be overwritten as 0s): 11 corresponds to unwritten memory and 00 corresponds to an invalid memory that has been tampered with.

Before we formally specify and verify the proposed solution, we briefly note that there is a real pragmatic need for *tamper-evident* rather than *tamper-proof* writeable storage. The *tamper-proof* requirement can be met only by some security mechanism ensuring authorised-only update of the vote store. This security mechanism would probably be implemented as some combination of physical constraints, together with hardware and software checks. It would most likely involve some complex encryption technique and it is not clear whether one could, or should, expect voters to trust such a complex system. Contrastingly, guaranteeing the *tamper-evident* requirement is a much simpler problem that — if done well — could be both trustworthy and trusted.

Implementing storage using a write-once data store has many obvious advantages when we consider tampering: obviously, any vote that has already been written cannot be overwritten? In fact, without a more formal model of the store, this is not guaranteed to be true. For example, one form of write-once storage could allow the flipping of an initial bit state to be done once and once only. This does not necessarily guarantee that a recorded vote cannot be overwritten as individual bits of a vote will not have been flipped when a vote is recorded. In fact, as with all storage mechanisms, the (encoding) protocol used for writing information to such a store will be the deciding factor in whether the tampering requirements are met. Furthermore, there are many reasonable variations of the tampering requirement. Without a precise statement, it is not clear whether we will be able to verify whether a given system (the store properties, together with the encoding protocol) is correct.

The key property of the encoding that we shall model is that if any (sub)set of 1 bits in a stored codeword are flipped to 0s then the result is no longer a valid code word. We then wish to establish that anyone with read access to the voting store can detect an invalid memory state, where at least one codeword is invalid, and consequently any tampering after[4] the election has been completed. The verification of this safety property requires modelling of the write-once behaviour in the chosen PROM implementation (checking that 1s can be re-written as 0s but that 0s cannot be changed) in conjunction with the encoding mechanism. It also requires the use of a special *election over* bit (bit pair in PROM) to signify that the election is over, and which must be unset and untampered with for new votes to be recorded (otherwise anyone with access to the voting machine could add unauthorised votes after the election, an attack known as ballot stuffing). We chose not to include the *election-over* behaviour in the model presented in this paper.

We note that this system is not tamper proof: attackers with write access to the vote store can still invalidate the election by overwriting vote data. However, this attacks would be easily identified by procedures for validating the storage state during and after the vote.

The main advantages of doing this design formally, in event-B, are development oriented:

- an abstract model can be easily validated as correctly expressing the requirement,
- the actual design model can be constructed incrementally through refinement of the abstraction,
- the refinement process can continue through to modelling at very fine grain levels of detail that correspond to the chosen low-level implementation architecture,
- we can more easily reason about different variations and combinations of encodings and storage media, and
- we can analyse possible problems of integrating this requirement with other requirements of the e-voting system, in general, and the vote storage, in particular.

Thus, we are more likely to develop a trustworthy storage.

A secondary benefit arises when we consider the issue of how to build public trust in our formally developed trustworthy system. We argue that the *correct-by-construction* technique, embodied in a reusable design pattern, will become more and more trusted as it is used to develop more and more systems that prove themselves to be trustworthy. As a consequence, using such a standard technique (and associated tools) in constructing critical systems will increase confidence in the systems' correctness, from both the developers and the public users.

With tool support for automatically checking our verification proof we have another advantage: if our proof tool is trustworthy then the design is sure to be correct provided the property that we have established, in the initial abstract model, is an accurate statement of the high level requirement. To make this transparent to the users (voters) it is essential that an initial abstract model is easy to understand and validate, and that they have good reason not to mistrust our proof tool and techniques. Our design approach facilitates this type of openness and transparency.

3.2.4.3 Verifying quality of service
Vérification de la qualité de service

The next research carried out was part of an applied research project — "Sécurité et Audit du Vote Electronique" (SAVE) — funded by the

La recherche entreprise ensuite faisait partie d'un projet de recherche appliqué — "Sécurité et Audit du Vote Electronique" (SAVE) — fondé par l'Agence Na-

[4] It is trivial to extend our model to dynamically detect tampering during an election but for simplicity and conciseness we do not present details of this variation of tamper-evidence.

French *Agence National de la Recherche* (ANR), in which the objective was to develop a prototype for an innovative e-voting system for use in France. In our chosen system, we have two clear voting innovations (for France) that the system will be required to support. Firstly, we wish to allow voters to be able to go to any polling station in order to vote; currently they are required to go to a particular station. We refer to this as a *VoteAnywhere* feature. Secondly, we wish to allow voters to be able to re-vote so that a previously recorded vote is overwritten by a new vote; currently voters have no way of changing their vote in such a way. We refer to this as a *ReVote* feature.

Our main research contribution — published in [GLR08a] — was to demonstrate that it is possible to formalise the functional requirements (*VoteAnywhere* and *ReVote*) and to analyse, through simulation of formal models, the quality of service offered by different architectures for distributed e-voting systems (that meet these functional requirements).

tionale de la Recherche (ANR), dans laquelle l'objectif était de développer un prototype pour un système innovant de vote électronique à utiliser en France. Dans le système choisi (pour la France), nous avons deux innovations électorales que le système devra à supporter. Premièrement, nous souhaitons permettre aux votants d'avoir la possibilité d'aller à n'importe quelle bureau de vote pour voter ; actuellement ils sont obligés d'aller dans un bureau spécifique. Nous faisons référence à ceci comme un *feature "VoteAnywhere"*. Deuxièmement, nous souhaitons permettre aux votants de pouvoir revoter pour qu'un vote précédemment enregistré soit remplacé par un nouveau vote ; actuellement les votants n'ont aucun moyen de changer leur vote d'une telle manière. Nous faisons référence à ceci par le *feature de "ReVote"*.

Notre principale contribution de recherche — publiée dans [GLR08a] — était de démontrer qu'il était possible de formaliser les besoins fonctionnels (*VoteAnywhere* et *ReVote*) et d'analyser, à travers la simulation de modèles formels, la qualité de services offerts par différentes architectures pour les systèmes distribués de vote électronique (qui remplissent ces besoins fonctionnels).

For simulation and analysis of quality of service requirements we have chosen to use Estelle [DAC89], a Formal Description Technique standardised by ISO [ISO97]. Although this technique is not as popular as other better known formal methods, it is well suited to the analysis task outlined in our research. Its main application field is the formal specification of distributed systems using communication protocols, and it permits a clear split between the definition of the global architecture of the system and the internal behaviour of its components.

The most significant threat to being able to meet quality of service requirements is a complete denial of service, where users of the system are unable to execute core functionality. In the case of e-voting machines, such a denial of service would prohibit anyone from recording a vote. When an e-voting system relies on components that are accessed across a network then a significant denial of service threat would arise if the network was not reliable (or if it was not resistant to attacks). This is a well documented concern for remote voting, including internet voting. However, it is also a concern in our chosen system where the network is a key component.

The main advantage of our approach — reported in this paper — is the ability to reason about e-voting quality of service (including denial of service) early in the development process. Analysis of such issues should be done as soon as possible — which means verifying that any proposed high-level design (architecture) is able to meet the quality of service requirements. Making a choice between alternative architectures should not be done without having first completed a verification of such requirements.

In order to reason about quality of service properties in Estelle we are obliged to simulate behaviour of our

proposed architectures in different environments. The standard approach is to identify key environmental parameters and to analyse the system's performance as these parameters change. In our case, we have two key dynamic parameters: the distribution over time of voters attempting to vote and the distribution over time of network inoperability.

For voter distributions we had access to a large bank of data from which we were able to generate a typical distribution curve . For network operation, we had no data from which we could generate typical distribution curves of network downtime for voting systems. Consequently, we were obliged to use more generic information from our network providers about the types of distribution curves that their networks would offer under normal conditions (not associated with e-voting). Then, in order to stress test the system, we chose to align peak voting times with the peak network downtimes.

The key role of the simulation is to show that certain architectures will not be able to meet the quality of service requirements. The architectures that are verified to meet the requirements (through simulation) can progress for further development, but need to be more thoroughly analysed during later development steps.

Through our simulations, we have helped guide design decisions made by the e-voting system developers. By demonstrating, early in the design process, the inappropriatness of certain architectures (with respect to their inability to meet quality of service requirements when the underlying network is not perfect) we have significantly aided the design process. Furthermore, by providing formal models we are more confident that the final system will meet the innovative functional requirements of *VoteAnywhere* and *Revote*.

3.2.4.4 Feature Interactions and an e-voting software product line — Interactions de service et une ligne de produit logiciel

A significant number of failures in e-voting systems have arisen because of poorly specified requirements, combined with an ad-hoc approach to engineering multiple variations of similar machines.

In [GLR09] we demonstrate that e-voting is a suitable domain for leveraging state-of-the-art in software product line (SPL) engineering techniques and tools. We propose, based on examples of typical requirements, that a feature-oriented approach to e-voting domain analysis is a good foundation upon which to carry out commonality and variablity analysis. Simple analysis of our core and optional features (and their variants) leads us to believe that feature interactions are a major problem in voting systems. We conclude that a formal software product line would help to manage the composition of features in such a way as to eliminate interactions in the requirements models, before particular e-voting systems are instantiated.

Un nombre significatif d'échecs dans les systèmes de vote électronique est survenu à cause de besoins pauvrement spécifiés, combinés avec une approche ad hoc vers la construction de multiples variations de machines similaires.

Dans [GLR09], nous démontrons que le vote électronique est un domain exemplaire pour exercer une influence sur des techniques de construction et des outils ultramodernes dans les lignes de produits logiciel (*software product lines* (SPLs)). Nous proposons, le tout basé sur des exemples typiques de besoins, qu'une approche orientée service pour l'analyse de domaine du vote électronique est une bonne fondation sur laquelle mettre en œuvre une analyse de commonalité et de variabilité. Une simple analyse de notre noyau et des *features* optionnelles (et leurs variantes) nous conduit à croire que les interactions de services sont un problème majeur dans les systèmes de vote électronique. Nous concluons qu'une SPL formelle aiderait à gérer la composition de *features* de telle manière à éliminer les in-

teractions dans les modèles de besoins, avant que des systèmes de vote électronique particuliers soient créés.

Software Product Lines (SPLs) [CN02] are attracting attention in the area of applied software engineering research. The challenge, which this article addresses, is to demonstrate how and why an e-voting SPL could be built. E-voting systems correspond in terms of size and complexity to those reported in a number of SPL case studies [Bos99b]. The number of variations across systems [SP06] is large enough to merit an SPL approach, but not so large as to be unmanageable. Furthermore, these systems exhibit a large amount of common functionality and so the potential for re-use is high. The aspect of e-voting that may be more challenging is that the software may be considered (safety or mission) critical [MG03]. However, recent research suggests that SPLs can be used to develop safety critical systems [Liu07].

The paper notes that analyses of existing systems — such as the state of Ohio's EVEREST report[BEH+08] — have identified verification issues directly related to foundational concepts in software product lines: "parameterized families of components" [McI68], a "family of related programs" [Dij72, Par76], and "structuring commonality and variability according to features"[KCH+90].

An e-voting system has a myriad of layers of inter-related legal requirements to meet. Further, each voting system has to meet specific needs which are not directly addressed by the laws and standards. The requirements of the system must somehow integrate these specific needs with multiple layers of laws and standards. As changes are made to requirements within different layers, in parallel, then who is responsible for ensuring that the requirements can be re-integrated in a coherent manner?

In our research, we selected examples of interactions that illustrate the need for more formal modelling and analysis. First we considered the most challenging requirements integration problem in e-voting: how to ensure both anonymity and verifiability? Secondly, we analysed potential interactions between the two innovative features of our chosen system: re-voting anywhere. Finally, we discussed the interactions that arise when the innovative features of our chosen system have to integrate with an existing non-core feature common to French elections: procuration.

In all instances we considered quality of service (*QoS*) to be a core requirement, so that the time required to record an individual vote should never be "unreasonable"[GLR08a].

In conclusion, as we cannot guarantee the reliability of any non-local communication network, we currently reject any voting process where an elector depends on a non-local communication in order to be able to register their individual vote. This issue is critical in many of the feature interactions that we subsequently considered.

3.2.4.5 Formal modelling of e-voting system architectures — Modélisation formelle d'architectures de système de vote électronique

It is clear that the formal verification of e-voting system models would help to address problems associated with certification against standards, and would improve the trustworthiness of the final systems. However, it is not yet clear how best to carry out such formal modelling and verification in order to leverage the compositional nature of the problem, and manage the complexity

Il est clair que la vérification formelle des modèles de vote électronique aiderait à traiter des problèmes associés à une certification contre standards et améliorerait la fiabilité des systèmes finaux. Cependant, il n'est pas encore évident de connaitre comment mettre en œuvre une modélisation formelle et une vérification de la sorte de la meiilure faççon, ceci d'exploiter la nature qui compose le problème, et de gérer la complexité de la tâche.

of the task.

The choice of modelling language — for expressing the high level design and architecture of an e-voting system — poses many problems due to the complex mix of requirements that such a system is required to meet. Different modelling languages are more-or-less suited to the verification of different critical requirements. Thus, we reported in [GLR10] on a mixed model approach: where we address 3 different types of critical requirements using 3 different modelling languages and development strategies.

Le choix d'un langage de modélisation — pour exprimer la conception haut niveau et l'architecture d'un système de vote électronique — pose de nombreux problèmes qui sont dus au mélange complexe de besoins qu'un tel système doit remplir. Différents langages de modélisation sont plus ou moins adaptés pour vérifier les différentes exigences critiques. Ainsi, nous avons fait un compte rendu dans [GLR10] sur une approche de modèles mélangés ; où nous traitons 3 différents types de besoins critiques utilisant 3 langages de modélisation différents et des stratégies de développement.

Firstly, we reported on network quality-of-service issues that are analyzed through simulation models. Secondly, we report on functional correctness of a counting process that can be validated through algebraic techniques. Finally, we report on the use of formal refinement to reason about the correctness of design steps when adding detail to an architecture model. To conclude, we acknowledged the main problem that arises from such a mixed-model approach to architecture verification: how can we be sure that the different models are coherent when we integrate them in a final implementation?

The final prototype system — with security mechanisms built on to our specified architecture — is in the process of being tested (against functional requirements). Through these tests (which were independently developed from the requirements models) we can verify that the count is correct, and that the three main features — *VoteAnywhere*, *Revote* and *Procuration* interact as required. We have no formal verification that the encryption algorithms central to the security mechanisms are correctly implemented — but the developers are experienced in using these same algorithms in a large number of security-critical systems.

It is clear from analysis of our development approach that the integration of our formal models is ad-hoc. We believe that are advantages from using different formal models at different stages of the development. However, establishing a re-usable method that coherently integrates such a mix of approaches is future research.

3.2.5 Important Technical Contribution: Interfaces can be constructed correctly

The use of formal methods in HCI development continues to be an important, ongoing area of research. Most approaches are based upon an operational approach to specifying the interaction as a state machine (much like a communication protocol). The key problem is in abstracting such operational models to an abstract set of functional properties that the interface respects, and which correspond to user requirements. We advocate working in the opposite direction — we start with a specification of the highest level abstraction and refine towards a correct implementation. The utility of such an approach has always been questioned, but our formal development of an e-voting system interface provides an excellent example of the general applicability of a correct-by-construction approach to user interface development.

In this subsection we provide some additional technical details regarding this approach (for the interested reader). The original paper — *Refinement: a constructive approach to formal software design for a secure e-voting interface*[CGM07b] — is extended with more generic models whose refinement allows for correct instantiation of different types of interface for different voting schemes.

Following the traditional refinement process using the B language, we propose in Figure 5 a high level model (`AllVotesAtOnce`): it abstracts away from individual votes and button presses.

MODEL
$AllVotesAtOnce$
SETS
$ELECTOR; CAND$
CONSTANTS
nbc, x
PROPERTIES
$nbc = card(CAND)$
$x \in 1..nbc$
DEFINITIONS
$valid(v) \mathrel{\widehat{=}} \exists n \cdot (n \in x..nbc \land v^{-1} \in 1..n \rightarrowtail CAND))$
VARIABLES
$vote, nbv$
INVARIANT
$vote \in ELECTOR \leftrightarrow (CAND \times (1..nbc)) \land$
$\forall e \cdot (e \in \mathsf{dom}(vote) \Rightarrow valid(vote[\{e\}]) \land$
$nbv \in \mathbb{N} \land$
$nbv = card(\mathsf{dom}(vote))$
INITIALISATION
$vote := \emptyset \parallel nbv := 0$
EVENTS
Voting $\mathrel{\widehat{=}}$
begin

$$vote, nbv :| \left(\begin{array}{l} vote \in ELECTOR \leftrightarrow (CAND \times (1..nbc)) \land \\ \forall e \cdot (e \in \mathsf{dom}(vote) \Rightarrow valid(vote[\{e\}])) \land \\ nbv \in \mathbb{N} \land \\ nbv = card(\mathsf{dom}(vote)) \end{array} \right)$$

end

Figure 5: AllVotesAtOnce specified in B

There is only one event — `Voting` — which models the votes of all electors who came to vote in *one shot*. A valid complete vote will be refined, in the next model, into the property that every individual vote is valid. For readers unfamiliar with the B modelling language, we note that:

- $ELECTOR$ is the set of all electors and $CAND$ is the set of all candidates,
- $vote$ is the variable which contains votes of all the electors,
- nbv is the cardinal of the domain of $vote$ representing the total number of votes,

- x is the minimum number of candidate preferences that must be marked for the vote to be considered valid. Thus, x acts as the parameter in our generic model to be instantiated in order to provide a concrete instance. For example, if we instantiate x to be 1 then we have the original concrete model specified in [CGM06], which corresponds to the requirement for a valid vote in Irish parliamentary elections. Alternatively, if we instantiate the parameter x to take the value of the $NumberofCandidates$ then we have an example of full preferential voting as commonly used in Australian elections. Finally, if we instantiate x to a value in between these two extremes then we have an example of partial preferential voting (common to council elections) such as that used for the Tasmanian Legislative Council.

Instead of modelling all the votes in *one shot* as above, in our next step we choose to refine the abstract specification so that each individual vote is modelled using the event `One_Vote` in the model `EachVoteAtOnce` (see Figure 6). Without such a refinement a realisitic implementation cannot be constructed.

MODEL
$EachVoteAtOnce$
REFINES
$AllVotesAtOnce$
SET

$STATE = \{voting, finish\}$

VARIABLES

$vote, nbv,$
$st, elector, vt, cvt$
INVARIANT

$elector \subseteq ELECTOR \ \wedge$
$vt \in elector \leftrightarrow (CAND \times (1..nbc)) \ \wedge$
$\mathsf{dom}(vt) = elector \ \wedge$
$\forall e \cdot \ (e \in \mathsf{dom}(vt) \Rightarrow valid(vt[\{e\}]) \ \wedge$
$cvt \in \mathbb{N} \ \wedge$
$cvt = card(elector) \ \wedge$
$st \in STATE \ \wedge$
$(st = finish \Rightarrow \mathsf{dom}(vote) = elector)$
INITIALISATION

$vote := \emptyset \ || \ nbv := 0 \ ||$
$st := voting \ || \ elector := \emptyset \ ||$
$cvt := 0 \ || \ vt := \emptyset$

EVENTS

One_Vote $\widehat{=}$
any e, v, n **where**
$st = voting$
$e \in ELECTOR - elector$
$n \in x..nbc$
$v \in 1..n \rightarrowtail CAND$
then
$vt := vt \ \cup \ (\{e\} \times v^{-1}) \ ||$
$elector := elector \ \cup \ \{e\} \ ||$
$cvt := cvt + 1$
end;
Voting $\widehat{=}$
when
$st = voting$
then
$vote, nbv := vt, cvt \ ||$
$st := finish$
end
END

Figure 6: EachVoteAtOnce specified in B

In this more concrete `EachVoteAtOnce` model, $STATE$ is an enumeration set which contains two values: $voting$ to represent a voting system that is *open*, and $finish$ to represent a state when the voting is *closed*. The

variable st contains one of these values, $elector$ is the subset of $ELECTOR$ recording the electors who have already voted, vt is the variable which contains these votes, and cvt is its cardinal. All votes in vt are valid. The event `One_Vote` models the vote of a single elector in *one shot* (as a single event).

In the `Vote` model in Figure 7 an elector e (who has not already voted) votes for candidates by pressing on the corresponding buttons.

MODEL
$Vote$
REFINES
$EachVoteAtOnce$
SET
$One\_Voting = \{start, valid, no\_elec\}$
VARIABLES
$vote, nbv,$
$st, elector, vt, cvt$
e, n, v, sto

INVARIANT

$e \in ELECTOR \ \wedge$
$v \in \mathbb{N} \leftrightarrow CAND \ \wedge$
$n \in 0..nbc \ \wedge$
$n = card(\mathsf{ran}(v)) \ \wedge$
$sto \in One\_Voting \ \wedge$
$(sto \neq no\_elec \Rightarrow$
$v \in 1..n \rightarrowtail CAND) \ \wedge$
$(sto \neq no\_elec \Rightarrow$
$e \in ELECTOR - elector) \ \wedge$
$(st = voting \ \wedge \ sto = valid \Rightarrow n \geq x)$

INITIALISATION

$vote := \emptyset \ || \ nbv := 0 \ ||$
$st := voting \ || \ elector := \emptyset \ ||$
$cvt := 0 \ || \ vt := \emptyset \ ||$
$sto := no\_elec \ || \ e :\in ELECTOR \ ||$
$v := \emptyset \ || \ n := 0$

EVENTS
Start_vote $\widehat{=}$
any E **where**
$sto = no\_elec \ \wedge$
$E \in ELECTOR - elector$
then
$sto := start \ \ ||$
$e := E \ \ ||$
$n := 0 \ \ ||v := \emptyset$
end;
Button_cand $\widehat{=}$
any c **where**
$sto = start \ \wedge$
$c \in CAND - \mathsf{ran}(v)$
then
$n := n + 1 \ \ ||$
$v := v \ \cup \{n + 1 \mapsto c\}$
end;
Button_valid $\widehat{=}$
when
$sto = start \ \wedge \ n \geq x$
then
$sto := valid$
end;
One_Vote $\widehat{=}$
when
$st = voting \ \wedge \ sto = valid$
then
$vt := vt \ \cup \ (\{e\} \times v^{-1}) \ \ ||$
$elector := elector \ \cup \ \{e\} \ ||$
$cvt := cvt + 1 \ \ ||$
$sto := no\_elec$
end;
END

Figure 7: Vote specified in B

One_voting is an enumeration set which contains three values: no_elec when no electors have voted, $start$ when the a new elector e starts to vote and $valid$ when the elector e pushes the button to validate their vote. The variable sto contains one of these values. Variable e contains the current elector, vt is their current vote which is modified when a candidate button is pushed, and n is the preference value of the chosen candidate.

We remark that the guard of the event `Button_valid` requires that $n \geq x$ and so we are sure that when an elector pushes this button then the partial vote v v has the required number of preferences and so v is a valid vote.

Remark also that we have no condition on n in the guard of the event `Button_cand`. When a candidate is not in the codomain of v we are sure[5] that $n < nbc$.

The advantage of this approach is that each interface model refinement is relatively easy to verify using the interactive theorem prover. The complexity of the development is evaluated through the number of proof obligations generated for the validation of each model or refinement[6]. Among generated proof obligations, a large number of them are automatically discharged by the prover. In our simple case study, 61 proof obligations are automatically discharged, and 12 are interactively derived using the tool but with human help. A second aspect that should be taken into account is the distribution of proof obligations through the global process.

The first model is easy to prove since it is very abstract. The refinement model $EachVoteAtOnce$ requires two interactive proofs which are quite simple, and only one difficult proof obligation stating the existence of some n which is related to the validity of the vote. The last refinement model is much more complex and the longest proof (requiring inductive reasoning) is 34 steps long. Table 1 illustrates how the complexity of the proof process grows with each refinement of the model.

Model	Total Number proof obligations	Interactive proofs
AllVotesAtOnce	3	0
EachVoteAtOnce	16	2
Vote(+ Cand_vote)	42	10
TOTAL	61	12

Table 1: Proof Obligations

E-voting Bibliography

[BEH+08] Kevin Butler, William Enck, Harri Hursti, Stephen McLaughlin, Patrick Traynor, and Patrick McDaniel. Systemic Issues in the Hart InterCivic and Premier Voting Systems: Reflections on Project EVEREST. In *EVT'08: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2008 on Electronic Voting Technology Workshop*, Berkeley, CA, USA, July 2008. USENIX Association.

[Bos99] Jan Bosch. Product-line architectures in industry: a case study. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 544–554, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[CEB+05] Deirdre Carew, Chris Exton, Jim Buckley, Margaret McGaley, and J.Paul Gibson. Preliminary study to empirically investigate the comprehensibility of requirements specifications. In *Psychology of Programming Interest Group 17th annual workshop (PPIG)*, pages 182–202, University of Sussex, Brighton, UK, 2005.

[5] We have proven it thanks to the invariant property and the refinement construct. This proof, as with all others referred to in the paper, is available from the authors on request.

[6] The proof obligations for $Button\_cancel\_last\_cand$ and $Button\_cancel\_cand$ are both included in these counts.

[CGM06] Dominique Cansell, J. Paul Gibson, and Dominique Méry. Refinement: A constructive approach to formal software design for a secure e-voting interface. In A. Cerone and P. Curzon, editors, *Formal Methods for Interactive Systems (FMIS 2006)*, Macau SAR China, October 2006.

[CGM07a] Dominique Cansell, J. Paul Gibson, and Dominique Méry. Formal verification of tamper-evident storage for e-voting. In *SEFM*, pages 329–338. IEEE Computer Society, 2007.

[CGM07b] Dominique Cansell, J. Paul Gibson, and Dominique Méry. Refinement: A constructive approach to formal software design for a secure e-voting interface. *Electr. Notes Theor. Comput. Sci.*, 183:39–55, 2007.

[CN02] P. Clements and L. Northrop. *Software product lines*. Addison-Wesley Boston, 2002.

[DAC89] Michel Diaz, J. P. Ansart, and J. P. Couriat, editors. *Formal Description Technique Estelle: Results of the Esprit Sedos Project*. Elsevier Science Inc., New York, NY, USA, 1989.

[Dij72] Edsger W. Dijkstra. *Structured programming*, chapter Notes on structured programming, pages 1–82. Academic Press Ltd., London, UK, 1972.

[Gib05] J. Paul Gibson. E-voting requirements modelling: An algebraic specification approach (with cafeobj). Report NUIM-CS-TR-2005-14, Department of Computer Science, National University of Ireland, Maynooth., 2005.

[GJ09] J. Paul Gibson and Doug Jones, editors. *First International Workshop on Requirements Engineering for e-Voting Systems (RE-VOTE09)*, Atlanta, GA, USA, August 2009. IEEE.

[GLR08] J. Paul Gibson, Eric Lallet, and Jean-Luc Raffy. Analysis of a distributed e-voting system architecture against quality of service requirements. In Herwig Mannaert, Tadashi Ohta, Cosmin Dini, and Robert Pellerin, editors, *The Third International Conference on Software Engineering Advances (ICSEA 2008)*, pages 58–64, Sliema, Malta, October 2008. IEEE Computer Society.

[GLR09] J. Paul Gibson, Eric Lallet, and Jean-Luc Raffy. Feature interactions in a software product line for e-voting. In Nakamura and Reiff-Marganiec, editors, *Feature Interactions in Software and Communication Systems X*, pages 91–106, Lisbon, Portugal, June 2009. IOS Press.

[GLR10] J. Paul Gibson, Eric Lallet, and Jean-Luc Raffy. Engineering a distributed e-voting system architecture: Meeting critical requirements. In Holger Giese, editor, *Architecting Critical Systems, First International Symposium, ISARCS 2010, Prague, Czech Republic, June 23-25, 2010, Proceedings*, volume 6150 of *Lecture Notes in Computer Science*, pages 89–108. Springer, 2010.

[GM08] J. Paul Gibson and Margaret McGaley. Verification and maintenance of e-voting systems and standards. In Dan Remenyi, editor, *8th European Conference on e-Government*, pages 283–289. Academic Publishing International, July 2008. ISBN 978-1-906638-09-2.

[ISO97] ISO/IEC. Estelle: A formal description technique based on an extended state transition model. Technical Report ISO 9074, Information technology - Open Systems Interconnection, 1997.

[KCH+90] K.C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU-SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.

[KSRW04] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach. Analysis of an electronic voting system. In *IEEE Symposium on Security and Privacy (S&P04)*, pages 27–40. IEEE, 2004.

[Liu07] Jing Liu. Handling safety-related feature interaction in safety-critical product lines. In *ICSE Companion*, pages 85–86. IEEE Computer Society, 2007.

[McI68] D. McIlroy. Mass-produced software components. In *Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany*, pages 88–98, 1968.

[MG03] Margaret McGaley and J. Paul Gibson. E-voting: a safety critical system. Report NUIM-CS-TR-2003-2, Department of Computer Science, National University of Ireland, Maynooth, 2003.

[MG06] Margaret McGaley and J. Paul Gibson. A critical analysis of the council of europe recommendations on e-voting. In *EVT'06: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, pages 9–22, Berkeley, CA, USA, 2006. USENIX Association.

[MKSW06] D. Molnar, T. Kohno, N. Sastry, and D. Wagner. Tamper-evident, history-independent, subliminal-free data structures on prom storage — or — how to store ballots on a voting machine (extended abstract). *IEEE Symposium on Security and Privacy,*, 2006.

[Par76] David Lorge Parnas. On the design and development of program families. *IEEE Trans. Software Eng.*, 2(1):1–9, 1976.

[SP06] Krishna Sampigethaya and Radha Poovendran. A framework and taxonomy for comparison of electronic voting schemes. *Computers & Security*, 25(2):137–153, 2006.

[Sta85] William Stallings. *Data and computer communications*. Macmillan Publishing Co., Inc., Indianapolis, IN, USA, 1985.

3.3 Teaching Formal Methods — Enseigner les méthodes formelles

Education has a major role to play in software development becoming a real engineering *discipline*. Throughout my academic career, I emphasise the symbiosis between teaching and research. There are two key aspects. Firstly, one must try to bring ones research experience into the classroom. Secondly, one must also be aware of (and possibly carry out) research into innovative teaching techniques and practices. How best to edu-

L'éducation a un rôle majeur à jouer dans le développement logiciel qui devient une véritable discipline d'ingénierie. Tout au long de ma carrière universitaire, j'ai mis l'accent sur la symbiose existant entre l'enseignement et la recherche. Il y a deux aspects clés. Premièrement, nous devons essayer d'apporter notre expérience en matière de recherche au sein de la salle de classe. Deuxièmement, nous devons être également conscient (et peut-être mettre œuvre) de la

cate future engineers of software systems is a major challenge.

3.3.1 Learning From The Student

In 1998, formal methods were not being taught in all computer science/software engineering degree programmes. To address this problem we carried out research into a novel way of teaching formal methods: rather than concentrating on one particular method, we worked on a set of small case studies, using the mathematics in a flexible and intuitive manner, where the students could appreciate the need for formality[GM98]. Each case study was intended to illustrate, in turn, the need for some fundamental formalism. An unexpected result was that we also identified weaknesses in our understanding of formal methods: students' naive questioning helped us to identify how the methods, and the teaching of these methods, could be improved. In brief, it was not just the students who were learning!

Twelve years later, we continue to support the view that discrete mathematics is the foundation upon which software development can be lifted up to the heights of a true engineering discipline. The transfer of formal methods to industry cannot be expected to occur without first transferring, from academia to industry, graduates who are well grounded in such mathematical techniques. These graduates must bring a positive, yet realistic, view on the application of formal methods. Our goal is to produce software engineers who will go out into industry understanding the principles of specification, design and implementation. As these graduates develop their engineering skills, in an industrial setting, they should have the means, and the motivation, to integrate formality and rigour into any environment in which they are found. In this way, the formal methods should start to sell themselves.

recherche dans le domaine des pratiques et techniques d'enseignement innovatrices. Savoir comment éduquer de la meilleure façon possible des futurs ingénieurs en systèmes logiciels est un challenge majeur.

En 1998, les méthodes formelles n'étaient pas enseignées dans tous les programmes universitaires de sciences informatiques/génie logiciel. Pour traiter de ce problème, nous avons entrepris une recherche sur une nouvelle manière d'enseigner les méthodes formelles : plutôt que de se concentrer sur une méthode particulière, nous avons travaillé sur un ensemble de petites études de cas, en utilisant les mathématiques d'une manière flexible et intuitive, là où les étudiants pouvaient apprécier le besoin de formalité [GM98]. Chaque étude de cas avait pour objectif d'illustrer, tour à tour, le besoin de quelque formalisme fondamental. Le résultat fut inattendu nous avons aussi identifié les faiblesses dans notre compréhension des méthodes formelles ; le questionnement naïf des étudiants nous a aidé à identifier comment les méthodes et l'enseignement de ces méthodes, pouvaient être améliorés. En bref, les étudiants n'étaient pas les seuls à apprendre !

Douze ans plus tard, nous continuons à supporter l'avis que des mathématiques discrètes sont le fondement sur lequel le développement logiciel peut être rehaussé à la hauteur d'une véritable discipline d'ingénierie. On ne peut pas s'attendre à ce que le transfert des méthodes formelles vers l'industrie apparaisse sans qu'il y ait, au préalable, transfery du milieu universitaire à celui de l'industrie, des étudiants diplômés étant tout à fait à l'aise avec de telles techniques mathématiques. Ceux-ci doivent apporter une vue positive, mais néanmoins réaliste, sur l'application des méthodes formelles. Notre but est de produire des ingénieurs en génie logiciel qui iront dans l'industrie avec une compréhension des principes de spécification, de conception et d'implémentation. Comme ces diplômés développent leurs compétences en ingénierie dans un cadre industriel, ils devraient avoir les moyens, et la motivation d'intégrer formalité et rigueur au sein de tout environnement dans lequel

ils sont trouvés. De cette façon, les méthodes formelles devraient commencer à se vendre.

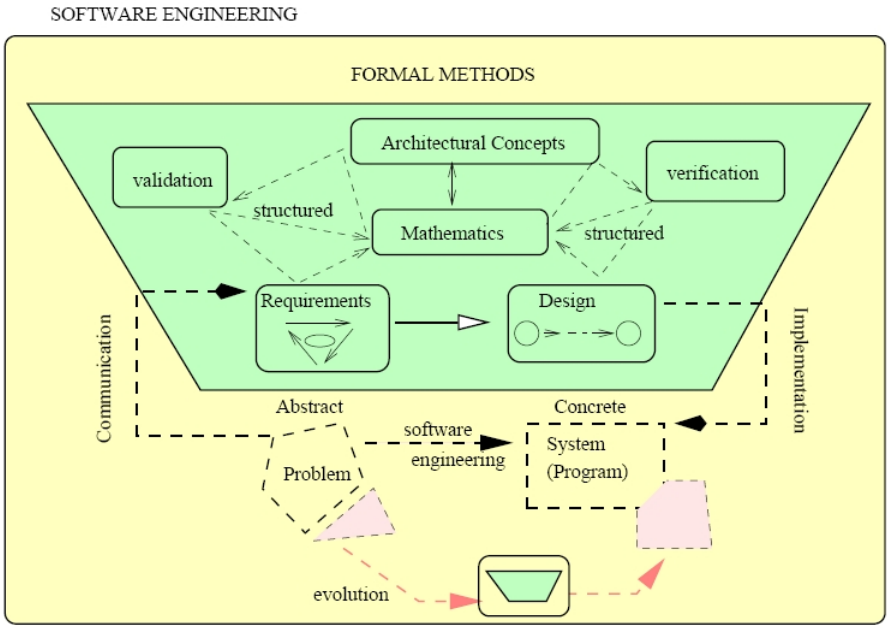

Figure 8: Software Engineering and Formality

Figure 8 illustrates the different steps in a traditional engineering process: analysis, requirements capture, design, implementation, and evolution. The formal methods are principally concerned with maintaining *correctness*, the property that an abstract model fulfils a set of well defined requirements [Bab87, Bol88, DeN87, Cus89], between the initial *customer oriented* requirements model and the final *implementation oriented* design. The formal boundaries break down at either end of the software development process because, in general, target implementation languages are not formally defined and customer understanding of their requirements is not complete.

Software development has reached the point where the complexity of the systems being modelled cannot be handled without a thorough understanding of underlying fundamental principles. Such understanding forms the basis of scientific theory as a rationale for software development techniques which are successful in practice. This scientific theory, as expressed in rigorous mathematical formalisms, must be transferred to the software development environment. Only then can the development of software systems be truly called *software engineering*: the application of techniques, based on mathematical theory, towards the construction of abstract machines as a means of solving well defined problems.

As a means of motivating the students, we mention a major study of the state-of-the-art in formal methods [CGR93], carried out 5 years before, which concluded by stating:

> " ...formal methods, while still immature in certain important respects, are beginning to be used seriously and successfully by industry to design and develop computer systems ... "

As the course advanced we came to see the value of the case studies. When introducing new concepts, a small

case study (often *toy* problems) were used to illustrate *why* formalism was needed, *what* sort of formalism could meet our needs and *how* to define and (re)use this formalism. This paper is 12 years old, but the simple problems have been re-used in many other courses, and have grown into more complex problems used in Problem Based Learning (PBL).

Perhaps the most important lesson that we, as lecturers, learned from the students was that abstraction was a very difficult concept. We introduced a case study based on sets, that has been re-used in a large number of courses and using a wide variety of formal languages. The original idea for this study came from a French text on graph algorithms [L94], where the author explained how the way in which sets where defined has a great influence in how they can be used for graph problems: where graphs are specified as sets of nodes and arcs. The goal of this case study was for them to see a development hierarchy as a step-by-step (refinement) process towards implementation.

Figure 9 illustrates the hierarchy which we examined.

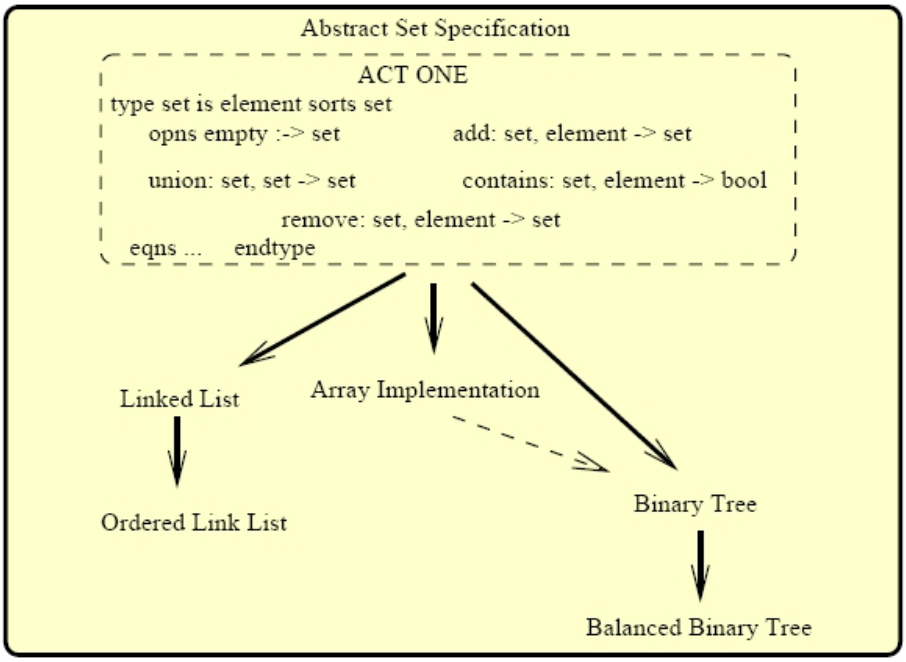

Figure 9: Designing a Set

This case study brought up some important fundamental issues (that we continued to see every time we taught the problem):

- Different teams quite naturally produce fundamentally two different (yet equivalent) specifications: group produce specifications in which `adding` an element first checked if the element was already in the set and did not change the set if this was true. Other groups produce specifications in which the `remove` is defined to remove multiple elements whilst the `add` allowed multiple entries. Other groups fall between these stools and did not realise that there is a problem with multiple elements. The students often wish to know which specification is best: here we have to explain the notion of *equivalence*, *invariants* and the need for *extensibilty*. A more difficult question is how to specify the set more abstractly so that both of these

specifications were *correct*.

- In the diagram, we see five different implementation strategies. Each arrow represents a design step in which the internal structure of the set is changed to *improve performance*. The question which I faced was:

  Why do we say that one model is more concrete (or abstract) than another if they are *equivalent*.

  Intuitively, there is something more concrete about a balanced binary tree than a linked list, but clearly from a purely functional point of view they provide the same behaviour. How do we formalise this notion of *abstraction level*?
- In the hierarchy diagram, there is a dotted link between the array implementation and the binary tree: it is *easy* to implement a binary tree using an array whilst it is not so *easy* to do this implementation with a linked list. The students wanted to know if this concept of *easy to implement using something* can be formalised.

After this case study we realised the need to look at the notion of *equivalence* in more detail, and the need to re-examine the notion of *abstraction level* from the point of view of *nondeterminism*.

Not much has changed since then except that we have many more examples of these issues in industry and we have much more better tools with which students can experiment with abstraction and nondeterminism.

3.3.2 Formal Requirements Engineering — Construire des besoins formels

In 2000, requirements engineering was being taught in most software engineering degree programmes around the world. However, most of these courses failed to teach the students how to formalise models and build abstractions. As a consequence, students were under the impression that requirements specifications were necessarily a combination of natural language text and informal diagrams.

To address this problem, we investigated how to integrate formal methods into a requirements engineering module[Gib00]. The paper reported on our experience in teaching requirements engineering using formal methods, where we advocated a multiple methods approach in which students get to evaluate a large range of specification languages: students are more likely to learn the principles of good requirements engineering rather than become experts in one particular (formal) method. The need for formality was introduced step-by- step, where new concepts were identified by the students through the use of case

En l'an 2000, l'ingénierie des besoins était enseignée dans la plupart des programmes universitaires de génie logiciel à travers le monde. Cependant, la plupart de ces cours ont échoué à enseigner aux étudiants comment formaliser des modèles et construire des abstractions. Comme conséquence, les étudiants avaient l'impression que les spécifications de besoins étaient nécessairement une combinaison de texte de langage naturel et de diagrammes informels.

Pour traiter de ce problème, nous avons enquêté sur comment intégrer des méthodes formelles dans un "requirements engineering" module [Gib00]. L'essai a fait le rapport sur notre expérience dans l'enseignement de l'ingénierie des besoins, en utilisant les méthodes formelles, où nous recommandons une approche de méthodes multiples, dans laquelle des étudiants ont la possibilité d'évaluer un vaste éventail de langages de spécification. Les étudiants ont plus tendance à apprendre les principes de bons l'ingénierie des besoins plutôt que de devenir des experts dans une méthode (formelle) particulière. Le besoin de formalité a été introduit étape par étape, là où de nouveaux concepts ont été identifiés

studies. These concepts are then formalised in the most appropriate language or notation. Students are encouraged to question the need for formality — each requirements engineering method is a compromise and the use of formal models needs to be placed within the context of the choices that a requirements engineer has to make.

par les étudiants à travers l'utilisation d'études de cas. Ces concepts sont ensuite formalisés dans le langage ou la notation le/la plus approprié(e). Les étudiants sont encouragés à questionner le besoin de formalité ; chaque méthode de l'ingénierie des besoins est un compromis et l'utilisation de méthodes formelles nécessite d'être placée au sein d'un contexte de choix que l'ingénieur de besoins doit faire.

This paper examines the requirement engineering issues, which were the most difficult to explain to the students for a variety of reasons:

- **Moving from informal to formal** — the requirements document is the first point of reference in the software development process. The step of going from informal understanding of a problem to a (formal) recording of this understanding is very difficult to learn (and to teach).
- **Coping with changing needs** — it is the nature of requirements to change. Thus, students must learn how to develop techniques which are both flexible *and* incremental. This involves a deep understanding of the compromises that exist within modelling.
- **Working in different problem domains** — to build good requirements models one must have a good understanding of the problem domain which is being modelled. When teaching a requirements engineering course there is always a risk that one will end up teaching about problem domains rather than about requirements modelling. However, students must also learn that requirements modelling and analysis go hand-in-hand: if they work in well-understood domains then they will never learn the importance of the analysis.
- **The need for customer orientation** — one must not lose sight of the customer in the whole process.

The main result of our research was to identify the main learning objectives, as a set of questions that students should be able toanswer?

- **Why is requirements engineering important?**
  Analysis is the process of maximising *problem domain understanding*. Only through complete understanding can an analyst comprehend the responsibilities of a system. The modelling of these responsiblities is a natural way of expressing system requirements. The modelling process increases understanding. Once the model is sufficiently rich to express all that is needed, then the analysis is complete and design can begin.
- **Why is the customer important?**
  The simplest way for an analyst to increase understanding is through interaction with the customer. The customer may be one person, in which case the *Requirements Capture and Analysis* (RCA) process is much simplified; however, it is more likely that the customer is a group of clients, each with their own particular needs. These clients may be people, machines, or both. One of the main problems in dealing with a set of customers is that the inter-related set of requirements must be incorporated into one coherent framework. Each client must be able to validate his (or her) own needs irrespective of the other clients (unless of course these needs are contradictory).
- **Why can the process never be perfect?**
  Interaction with the customer is an example of informal communication. It is an important part of analysis

and, although it cannot be formalised, it is possible to add rigour to the process. A well-defined analysis method can help the communication process by reducing the amount of information an analyst needs to assimilate. By stating the type of information that is useful, it is possible to structure the communication process. Effective analysis is dependent on knowing the sort of information that is required, extracting it from the customer, and recording it in some coherent fashion.

- **Why is requirements engineering difficult?**
  The analysis model must be capable of fulfilling two very different needs. Firstly, it must be *customer oriented*, i.e. there must be a direct correspondence between the model and how the customer views the problem. Secondly, the model must be useful to designers. The system requirements must be easily extracted, and the structure of the problem domain must be visible for (potential) re-use in the solution domain. The easiest way in which a model can play this dual role is if the same underlying notions and principles are present in the problem and solution spaces.
- **How can formality help?**
  Mathematical rigour is necessary for formal validation, testing and completeness and consistency checking. The advantages of formal methods in the specification of requirements are well documented (see [BG81, BJA82, Dil90], for example).
- **How can formality hinder?**
  Formal methods do not come for free. They require much more rigorous development techniques which are more time consuming and more difficult to master. Furthermore, formal methods risk being too difficult for the client (or engineer) to understand. Extra work is required to make them presentable to anyone other than the (mathematically oriented) requirements engineer.
- **What is the difference between validation and verification?**
  It is important that the requirements engineer understands that validation is about checking that a formal model correctly captures the client's needs, and that verification is about checking that a formal model meets the requirements of another formal model. We can verify the consistency of a requirements model by showing that it's operational requirements meet its logical requirements, but this is not validation.

We have reused this list of questions in every subsequent module that we have taught which includes requirements engineering as a topic.

3.3.3 Correct Design
— La conception correcte

Teaching software engineering students about design is very challenging. In general, students will learn about design through a module teaching a graphical modelling language. Our experience shows that this can result in students learning how to represent and comprehend designs but having very little understanding of design as a process. When reviewing design artefacts, students often ask whether the designs are *good*. This leads to the realisation that there is lack of understanding

Enseigner la conception ("design") à des étudiants en génie logiciel comporte un grand challenge. En général, les étudiants apprenent sur la conception, par un modèle enseignant le langage de modélisation graphique. Notre expérience montre que ceci a comme résultat le fait que les étudiants apprennent comment représenter et comprendre des *designs* tout en ayant peu de compréhension du *design* comme processus. Lorsqu'on reconsidère des objets (artefacts) de *design*, souvent les étudiants se demandent si les *designs* sont

of the fundamental question of whether a design can be said to be *correct*.

Of course, the notion of *correctness* will generally be covered by another module, typically called "formal methods". Unfortunately, our experience also shows that formal methods courses can lead to students learning how to build formal models — much like they would build programs — without achieving a good understanding of non-determinism and abstraction; and without seeing how formal methods can help in the process of design. We argued, in [GLR08b] that the teaching of software design needs to be better integrated with the teaching of formal methods.

bons. Ceci conduit à la réalisation qu'il y a un manque de compréhension de la question fondamentale à savoir si un *design* peut être dit comme "correct".

Bien sûr, la notion de correction se couvrira généralement par un autre module, typiquement appelé "méthodes formelles". Malheureusement, notre expérience montre aussi que les cours de méthodes formelles peuvent conduire à l'apprentissage des étudiants sur la façon de construire des méthodes formelles — comme ils construiraient des programmes — sans acquérir une bonne compréhension du non-déterminisme et de l'abstraction ; et sans voir comment les méthodes formelles peuvent aider dans le processus de *design*. Nous avons démontré, dans [GLR08b], que l'enseignement de *design* de génie logiciel a besoin d'être mieux intégré dans l'enseignement des méthodes formelles.

One of the least well understood aspects of software development is the role of design in bridging the gap between *what* (requirements) and *how* (implementation). Inexperienced software designers fail to treat design as a process, and as a consequence become experts in representing the (static) artefacts using models/languages but fail to master the evolution of design.

During the transition from procedural to object-oriented programming languages, there was a realisation that the boundary between design and implementation was becoming even more blurred. In a controversial article in the C++ Journal, Reeves[Ree92], stated: "...about ten years ago I came to the conclusion that, as an industry, we do not understand what a software design really is. I am even more convinced of this today." Reeves goes on to argue that considering the source code as being the design overcomes one of the fundamental issues associated with software design: how can we be sure that it will work correctly?

> "...when real engineers get through with a design, no matter how complex, they are pretty sure it will work. They are also pretty sure it can be built using accepted construction techniques. In order for this to happen, hardware engineers spend a considerable amount of time validating and refining their designs."

These ideas have much more resonance when we consider recent growth in agile development[Mar02].

In fact, the notion that the design is not finished until it has been coded and tested is not, as it would seem at first sight, at odds to a formal approach to software design. In a formal approach, designs are coded (using formal specification languages) and they are tested and refined. Unfortunately, teaching formal methods to software engineers is no guarantee that they will use them during design! Ken Robinson[Rob04] identifies a clear problem with the teaching of formal methods: "It is frequently the case that the other courses make no reference to, or use of, the formal techniques studied in the Formal Methods course."

We argued that it is the responsibility of the teachers of formal methods to incorporate aspects of *all* other software engineering courses in their teaching (not just design). However, the main contriution of our work was in the integration of formal methods and design, with specific examples given using UML[Boo99].

The example problem that has had most impact on our teaching of design considered the implementation of a queue (of FIFO behaviour) using stacks (of LIFO behaviour). We specify the requirements as a Queue of integers[7] and state that the students must implement the FIFO behaviour using only two integer Stacks (LIFO behaviour) to store the queue contents. As a design exercise, students typically adopt 1 of 2 options:

- Design1: The queue is specified as having two stack components — which we will name as a `pushstack` and a `popstack`. When a push request is made of the queue then this element is pushed directly onto the `pushstack`.When a pop request is made of the queue then move all elements from the `pushstack` on to the `popstack` then pop off the last element of the `popstack` and then move all the elements back on to the `pushstack`.
- Design2: The queue is has two stack components — which we will name as a `mainstack` and a `tempstack` — and a boolean representing whether or not the `mainstack` is `ready to push`. (If it is not `ready to push` then we say that it is `ready to pop`). When `ready to push`[8]: if a push request is made of the queue then this element is pushed directly onto the `mainstack`, if a pop is requested then all the elements are moved from the `mainstack` to the `tempstack`, the `mainstack` and `tempstack` are swapped, the state is changed to `ready to pop` and the element popped off the `mainstack`.

At this stage we ask the students to evaluate the quality of their designs. Most students identify the following inter-related design quality criteria: simplicity, understandability, implementability, extensibility, modularity, maintainability, re-usability, efficiency (time and memory), robustness and reliability. In our experience students will ask about the *correctness* of their design only if they have already studied formal methods. When asked if the design will work, most students reply that they will test their implementation to make sure that it does.

Analysis of Design1 and Design2 usually leads to students identifying that Design1 is easier to understand and implement, but that Design2 may be more efficient. Representing the two designs in UML often leads to the students realising that the two designs appear to be structurally the same, but quite different in terms of their dynamic behaviour. The class diagram, in figure 10, illustrates that using UML leads to further investigation of design alternatives that are not so obvious from working only with a formal modelling language.

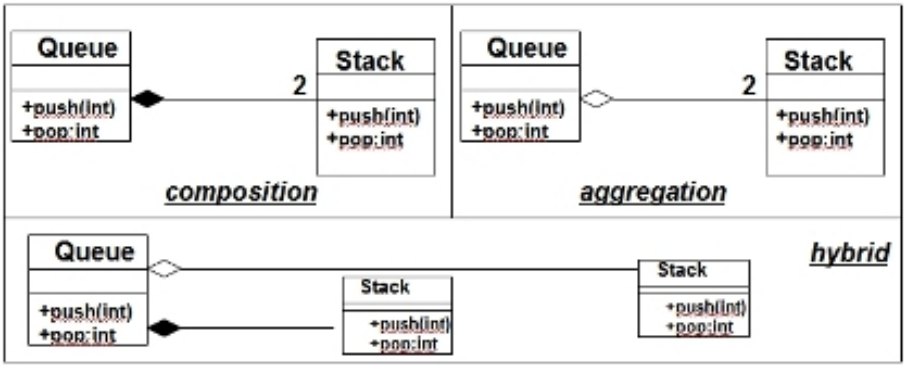

Figure 10: Aggregation or Composition?

[7]Note that this problem takes on a different nature if we allow the modelling of parametric classes of behaviour.
[8]The `ready to pop` case can be treated similarly.

Most students choose to model the assocation between the Queue and its class components as composition. The remaining students usually model this using aggregation. We ask the question as to why a hybrid model (using both composition and aggregation) is, in general, never considered.

We then ask the students to argue (demonstrate) whether the designs are correct before they implement them. Typically, they are not able to convince themselves that the designs are correct but they do identify unsafe states of the designs that should not arise. We then show them how these can be modelled using invariants. For example, in Design1 the Queue system is unsafe if the `popstack` is not empty when an element is pushed on to the `pushstack`.

We have followed two different routes from this point. Firstly, students can implement their designs (typically in Java). Secondly, students formalise their designs and attempt to prove that the designs are correct. In the first instance, student implementations often do not meet the queue requirements (which can be found through testing): students then need to discuss whether this signifies that their designs are incorrect. In the second instance, students usually manage to model the abstract queue requirements, but fail to see how they can refine their queue into two communicating stacks. The best students manage to model the designs formally but fail to prove the refinement relation (and hence the correctness). However, when asked to implement their formal designs (again, in Java) they usually do not make the same programming errors.

3.3.4 Starting Young
Commencer jeune

In many countries around the world, there is a crisis in the teaching of mathematics and computer science. Governments have tried to address the problem by investing in computers in schools; when they should have invested in teaching computer science in schools. Formal methods bridge the boundary between computing and mathematics in a natural way. Through our experience of teaching algorithmic thinking in schools, young children have been observed using concepts such as refinement, proof, abstraction, complexity, nondeterminism, equivalence, etc. . . in their own reasoning about problems. We argue that this ability needs to be better leveraged in order to improve both the teaching of mathematics but also to improve childrens' understanding of computer science as a discipline in its own right.

Whilst lecturing in Ireland and France, we have tried to convince colleagues that formal methods should be taught in the first year of undergraduate degree programmes. In general, this proposal is not taken seriously as they argue that

Dans de nombreux pays à travers le monde, il existe une crise dans l'enseignement des mathématiques et de l'informatique. Les gouvernements ont tenté de traiter le problème en faisant l'investissement d'ordinateurs à l'école, alors qu'ils auraient dû investir en enseignant les sciences informatiques à l'école. Les méthodes formelles réduisent l'écart entre l'informatique et les mathématiques d'une façon naturelle. A travers notre expérience de l'enseignement de la pensée algorithmique dans les écoles, nous avons observé la façon dont de jeunes enfants, utilisant des concepts tels que raffinement, preuve, abstraction, complexité, nondéterminism, équivalence, etc., ont leur propre raisonnement sur des problèmes. Nous avançons que cette capacité nécessite d'être mieux exploité afin d'améliorer à la fois l'enseignement des mathématiques mais aussi améliorer la compréhension des enfants sur l'informatique comme discipline à part entière.

Tout en enseignant en université en Irlande et en France, nous avons tenté de convaincre les collègues que les méthodes formelles devraient être enseignées dans les programmes universitaires de premier et de deux-

the students are not ready for the complicated mathematics. In order to address this problem, we carried out a number of experiments with school children in order to establish that formal methods could be taught before the students come to University. The results of these experiments were later published[Gib08a].

ième cycles. En général, cette proposition n'est pas prise sérieusement car ils avancent que les étudiants ne sont pas prêts pour des mathématiques compliquées. Afin de s'occuper de ce problème, nous avons mis en oeuvre un certain nombre d'expériences avec des écoliers pour établir que les méthodes formelles pourraient être enseignées avant que les étudiants ne parviennent à l'université. Les résultats de ces expériences ont été publiés ultérieurement [Gib08a].

This research paper reports on the teaching of formal methods to children as young as seven years old. It supports the well-established view that there are advantages in teaching mathematics constructively. For example Clements stated in 1990[CB82]: "Educational research offers compelling evidence that students learn mathematics well only when they construct their own mathematical understanding." The work reported is motivated by our own experiences in teaching computer science to young children: Java programming[Gib03], the Computer Science Unplugged Tutorials[Bel00], and problem based learning (PBL)[OG05a]. It builds on research that suggests that formal software engineering concepts form the basis for how children learn to solve problems[Gib05].

In "How to Solve It: A New Aspect of Mathematical Method", Polya[Pol71] states: "A good teacher should understand and impress on his students the view that no problem whatever is completely exhausted." Our approach has been to take problems that one would normally see in university and to rework them for school children. The children are then encouraged to take the problems in whatever direction they wish, and as far as they want. In this way, young children quickly identify fundamental concepts in computer science.

This paper reports on some of the case studies with which we have had most success. We do not claim to have carried out a verifiable educational experiment: we report on observations that have been made over a number of years through interaction with hundreds of children (aged 7 to 18).

In each of the sessions that we run in the schools, we are mindful that our goal is that the children learn fundamental concepts. Following the PBL philosophy, we do not explicitly teach the children about these concepts; we help the children to discover them through interaction with a specific problem. Each problem has the goal of the students discovering at least one of the following:

- *Proof* — the evidence or argument that compels one to accept an assertion as true.
- *Theorem* — a proposition that is true in all cases.
- *Conjecture* — an unproven proposition for which there is some sort of empirical evidence.
- *Constructive proof* — demonstrates the existence of a mathematical object with certain properties by giving a method (algorithm) for creating such an object.
- *Algorithm* — any procedure involving a series of steps that is used to find the solution to a specific problem.
- *A deterministic algorithm:* behaves predictably. Given a particular input, it will always produce the same output, and the underlying machine will always pass through the same sequence of states.
- *Correctness* — an algorithm can be proven to be correct with respect to a specification.
- *Refinement* — the verifiable transformation of an abstract (high-level) formal specification into a concrete (low-level) executable program. It guarantees the correctness of the program by construction.
- *Invariant* — an expression whose value does not change during algorithm execution; which can implement a required safety property in the specification

- *Computational Complexity* — the scaleability of algorithms with respect to the use of resources (typically time and space).

It is beyond the scope of this work to analyse all of these learning objectives, and to demonstrate how our sessions help children to reach them. Rather, we gave examples of sessions that we run with the youngest children (aged seven to nine). These illustrate how the formal methods concepts arise quite naturally out of the games that we play.

The main contribution of this work was to demonstrate that it is possible to teach young children formal methods concepts through games and problem based learning. Children who are disinterested in mathematics regain an interest when they see that mathematical modelling can help them reason about algorithms and games. We do not make any claims about whether our sessions improve the childrens' mathematical ability. However, we do believe that this type of session is a good way of introducing computer science in schools. (We have also demonstrated that this problem-based learning approach can be used to teach university students[OG05a] about computer science.)

As formal methods are foundational to computer science, it should not be any surprise that the rigorous, mathematical, analysis of algorithms and computations should be a major part of teaching computer science to children. This is not a new idea — it was discussed at the TFM workshop in Ghent in 2004[DB04], where daRosa presented research into the teaching of recursive algorithms as part of a high school mathematics course[dR04]. We already know that computer science education has need of mathematics; perhaps now the mathematics teachers can be persuaded to consider computer science (formal methods) as a good way of teaching mathematics.

3.3.5 Weaving Formal Methods — Tisser des méthodes formelles

The idea of weaving formal methods through computing (or software engineering) degrees is not a new one. However, there has been little success in developing and implementing such a curriculum. Formal methods continue to be taught as stand-alone modules and students, in general, fail to see how fundamental these methods are to the engineering of software. A major problem is one of motivation — how can the students be expected to enthusiastically embrace a challenging subject when the learning benefits, beyond passing an exam and achieving curriculum credits, are not clear? Problem-based learning has gradually moved from being an innovative pedagogic technique, commonly used to better-motivate students, to being widely adopted in the teaching of many different disciplines, including computer science and software engineering. Our experience shows that a good problem can be re-used

L'idée de tisser des méthodes formelles au travers de diplômes en informatique (ou génie logiciel) n'est pas nouvelle. Néanmoins, il y a eu peu de succès dans le développement et l'implémentation d'un tel programme d'études universitaires. Les méthodes formelles continuent à être enseignées comme des modules autonomes et les étudiants, en général, échouent à voir comment ces méthodes sont fondamentales pour la construction d'un logiciel. Un problème majeur est celui de la motivation : comment peut-on attendre des étudiants qu'ils adoptent avec enthousiasme un sujet difficile alors que les bénéfices d'apprentissage (au-delà du fait de réussir à un examen et d'acquérir des crédits du programme d'études universitaires) ne sont pas clairs ? L'apprentissage par problèmes est graduellement passé comme étant une technique pédagogique innovante — utilisée habituellement pour mieux motiver les étudiants — à une pédagogie largement adoptée dans l'enseignement de nombreuses disciplines différentes, incluant l'informatique et le génie

throughout a student's academic life.

In fact, the best computing problems can be used with children (young and old), undergraduates and postgraduates. In [Gib08b] we presented a process for weaving formal methods through a University curriculum that is founded on the application of problem-based learning and a library of good software engineering problems, where students learn about formal methods without sitting a traditional formal methods module. The process of constructing good problems and integrating them into the curriculum is shown to be analagous to the process of engineering software. This approach is not intended to replace more traditional formal methods modules: it will better prepare students for such specialised modules and ensure that all students have an understanding and appreciation for formal methods even if they do not go on to specialise in them.

logiciel. Notre expérience montre qu'un bon problème peut être réutilisé à travers la vie universitaire d'un étudiant.

En fait, les meilleurs problèmes informatiques peuvent être utilisés avec les enfants (petits et grands), avec les étudiants de premier et de deuxième cycle, et avec les étudiants de troisième cycle. Dans [Gib08b], nous avons présenté un processus pour tisser des méthodes formelles à travers un programme d'études universitaires qui est fondé sur l'application d'un apprentissage par problèmes et d'une bibliothèque composée de bons problèmes de génie logiciel, où les étudiants apprennent sur les méthodes formelles sans se présenter à un module traditionnel de méthodes formelles. Le processus pour construire de bons problèmes et pour les intégrer dans un programme d'études est montré comme étant analogue au processus du génie logiciel. Cette approche n'a pas pour intention de remplacer des modules plus traditionnels de méthodes formelles : elle préparera mieux les étudiants pour de tels modules spécialisés et assurera que tous les étudiants ont une compréhension et une appréciation des méthodes formelles même s'ils ne se spécialisent pas dans ce domaine.

Parnas makes a strong case that "Software Engineering Programmes are not Computer Science Programmes" [Par98]. He discusses the differences between traditional computer science programmes and most engineering programmes and argues that we need software engineering programmes that follow the traditional engineering approach to professional education. He summarises the issue as follows:

> "Just as the scientific basis of electrical engineering is primarily physics, the scientific basis of software engineering is primarily computer science. Attempts to distinguish two separate bodies of knowledge will lead to confusion. ... Recognising that the two programmes would share much of their core material will help us to understand the real differences."

Future scientists will add to our "knowledge base" while future engineers will design trustworthy products. His position is that: "engineers learn science plus the methods needed to apply science".

However, we must now ask where formal methods fit into this pedagogic structure and whether our approach to teaching formal methods should change depending on our target audience: computer scientists or software engineers. In our approach we see formal methods as the main bridge between computer science and software engineering. Without formal methods software engineering is not a true engineering discipline; and without formal methods computer science remains a mainly theoretical subject. Thus, teaching formal methods should not be seen as a problem to be solved; but it should be viewed as the answer to the fundamental question of how we can better educate computer scientists *and* software engineers.

Our problem based learning approach helps us to better adapt our teaching to our target audience. Our experience suggests that good problems are not good for only one type of students (engineering or science, or even arts and humanities). The best problems can be introduced to any of these students and through interacting with the problem (and with the guidance of the lecturer) the problem will dynamically evolve in order for particular learning objectives to be met. In general, engineering students will learn by trying to build solutions to the problems whilst science students will learn through trying to analyse them. Of course, the lecturer will be responsible for making sure that the students learn that these are complementary approaches and for finding the right balance for the particular type of student that is being taught.

We proposed that each problem should be set up to meet a specific curriculum objective. Each problem would then have a life-cycle similar to that seen for software and services, with key stages being specification, design, implementation, testing and maintenance. Once a problem is meeting a specific objective then it can be refined to incorporate other objectives. These objectives may be the responsibility of a single lecturer as part of a single module; but good problems will evolve to survive across different modules. In our experience this is most likely to happen when a single lecturer is responsible for multiple modules (where problems can be shared). However, in order to better weave our formal methods objectives through the curriculum we have to be able to also work with colleagues who do not teach formal methods but do teach other CS&SE modules.

We then proposed four complementary approaches to this weaving process. Firstly, look at the problems that are being used in other modules and incorporate them into a dedicated formal methods module. Secondly, offer to extend such problems (to meet the formal methods objectives) as part of the original modules in which the problems were taught. Thirdly, offer to extend your existing formal methods problems so that they incorporate learning objectives of colleagues teaching other modules. Fourthly, invite colleagues to participate in the PBL teaching in your own formal methods module(s).

We note that this integration should probably be done in an incremental fashion as we may end up replicating the feature interaction problem[Gib97] at the level of the requirements (learning objectives). To extend our analogy of a problem as being a service, with additional learning objectives as features, we can consider the curriculum to be a system of collaborating services. As our curriculum evolves we maintain the system by updating our problem set: adding new problems, removing unsuccessful problems and evolving successful problems. As with large, complex, software systems the best way to manage this process is to have a clearly documented set of requirements and procedures in place to map these requirements through to the final implemented system (via the design).

We concluded that the underlying architecture of the curriculum should be service-oriented in the sense that the main structure should support the evolution of the underlying objectives and the problems that are used to meet these objectives.

3.3.6 Important Technical Contribution: Formal Design Pattern

There is currently much research in the area of formal design patterns. During our teaching of formal object oriented development of software, we discovered a pattern that students had used across a number of different problems and projects. The pattern helped them to link:

- Event-B specifications,
- UML designs,
- Java implementation code, and

- Unit tests

In this subsection we provide some additional technical details (for the interested reader). The pattern is further described in the paper *Formal Object Oriented Development of a Voting System Test Oracle*[GLR11], where we applied it in our own research. This pattern is an excellent example of the symbiotic nature of research and teaching. In figure 11 we illustrate how the UML, Java and Event-B are loosely integrated in the design pattern.

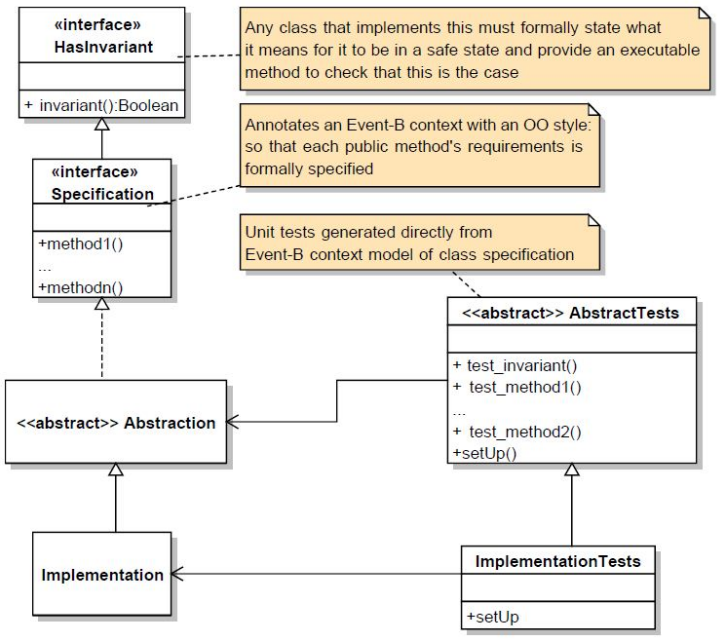

Figure 11: The generic development design pattern for all classes in the UML high-level design

The natural language requirements are used to identify the classes, their responsibilities, and the relations between them. These are then documented in a UML class diagram. Every class is then developed using the generic development design pattern:

- Class responsibilites are specified using an interface specification, i.e a set of methods whose functional requirements are formalised in an Event-B class model — a context written using an OO style.
- Every class must also specify an invariant method which states — in terms of the public responsibilities of the class — when class instances are in a safe state.
- For classes with a subset of external responsibilities which can be implemented abstractly through use of methods defined in the interface specification, these implementations (methods) are grouped together in a single abstract class for re-use.
- Every class has a set of unit tests whose code is derived directly from the Event-B specification. These tests are implemented abstractly (calling the abstract methods of the class abstraction) in an AbstractTests class.
- Any class that claims to correctly implement the abstraction must pass all the abstract unit tests.

Event-B contexts can be used to model abstract data types in an algebraic style. Following the formal object oriented approach to requirements modelling as advocated by Gibson[77], we address 3 key issues when modelling a class of behaviour:

- **A means of categorising entities into classes of behaviour:** We use Event B contexts with sets representing classes.
- **A means of recording the external interface of a class so that all methods can be statically 'type checked' for correctness:** We define constant operation signatures in the Event-B.
- **A means of defining the behaviour associated with each operation:** We, quite naturally, use Event-B axioms.

Three different types of methods/signatures — as proposed in [77] — are supported in our Event-B modelling style when specifying the behaviour of a UML `class`:

- *ACCESSORS:* methods with return values but which do not change the state of the object being called have signatures of the form:
  `method(p1, ... pn): result.`
  These are modeled in Event-B using an axiom of the form:
  $method \in class \times p1 \times \ldots \times pn \mapsto result$
- *TRANSFORMERS:* methods with no return values but which do change the state of the object being called have signatures of the form:
  `method(p1, ... pn).`
  These are modeled in Event-B using an axiom of the form:
  $method \in class \times p1 \times \ldots \times pn \mapsto class$
- *DUALS:* methods with return values that also change the state of the object being called have signatures of the form:
  `method(p1, ... pn): result.`
  These are modeled in Event-B using an axiom of the form:
  $method \in class \times p1 \times \ldots \times pn \mapsto class \times result$

When the behavioural requirements are specified only a subset of parameter values then the method relations are specified using partial functions; otherwise the functions are complete. (In our Java implementation we chose to implement such cases using exceptions.)

Our lightweight approach uses Event-B contexts to specify abstract class requirements, and facilitates modelling more concrete design and implementation details using Event-B machines, and their refinements. The move from requirements to design can be automated through the generation of an abstract Event-B machine corresponding to a class as specified in a context. This is similar to the approach in [77], which incorporates an ADT specification of a class of behaviour in a process algebra design through instantiation of generic parameters.

Once the Event-B specifications have been completed, we instantiate the abstract design pattern with the concrete classes in our initial design. In the following, we focus on the two classes which represent a Ballot at the interface and in the memory module (see figure 12).

It should be noted that the BallotAbstraction specifies requirements in terms of the public methods that must be implemented:

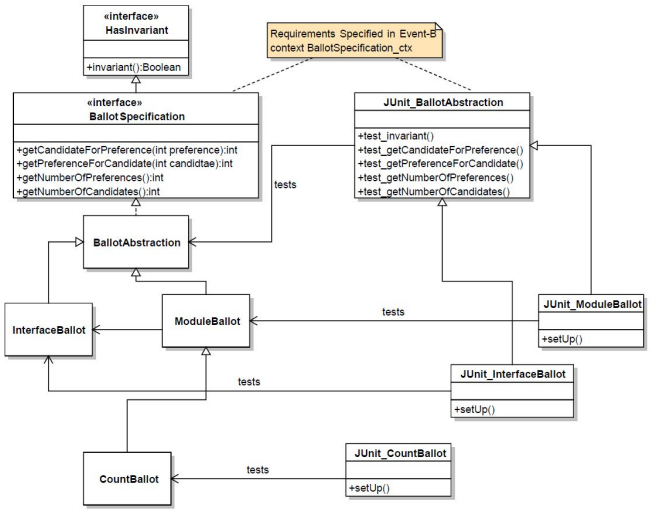

Figure 12: Instantiation of generic development design pattern for Ballots

- `getNumberOfCandidates`,
- `getNumberOfPreferences`,
- `getCandidateForPreference(int preference)`,
- `getPreferenceForCandidate(int candidate)`, and
- `invariant()`.

The Event-B axioms for the signature of the abstract ballot class are given in figure 13.

axm1 : $getNumberOfCandidates \in Ballots \rightarrow MIN\_NUMBER\_OF\_CANDIDATES \,..\, MAX\_NUMBER\_OF\_CANDIDATES$
Ballot Method Signature: Accessor

axm2 : $getNumberOfPreferences \in Ballots \rightarrow 1 \,..\, MAX\_NUMBER\_OF\_CANDIDATES$
Ballot Method Signature: Accessor

axm3 : $getPreferenceForCandidate \in (Ballots \times (1 \,..\, MAX\_NUMBER\_OF\_CANDIDATES)) \nrightarrow 0 \,..\, MIN\_NUMBER\_OF\_CANDIDATES$
Ballot Method Signature: Accessor

axm4 : $getCandidateForPreference \in (Ballots \times (1 \,..\, MAX\_NUMBER\_OF\_CANDIDATES)) \nrightarrow 0 \,..\, MAX\_NUMBER\_OF\_CANDIDATES$
Ballot Method Signature: Accessor

axm5 : $invariant \in Ballots \rightarrow BOOL$
Ballot Method Signature: Accessor

Figure 13: Abstract Ballot Signature Axioms

Note that within the domain of ballots, we must consider the count process in order to find a *TRANSFORMER* method. The `nextPreference()` method transfers a ballot to another candidate during tabulation. The proof that this method respects the abstract invariant is trivial (since the only state attribute that is changed is `currentPreference`) and is discharged automatically by the RODIN tool when modelled using Event-B.

The next step is to define the invariant requirements using Event-B axioms. This is illustrate in figure 14, where the invariant has been decomposed into the conjunction of seven requirements, of which we show four.

axm18 : $\forall b1 \cdot b1 \in Ballots \Rightarrow$
$((inv\_1(b1) = TRUE) \Leftrightarrow$
$getNumberOfCandidates(b1) \in$
$MIN\_NUMBER\_OF\_CANDIDATES \mathinner{..} MAX\_NUMBER\_OF\_CANDIDATES)$
inv_1 requirement: the number of candidates is in range: MIN_NUMBER_OF_CANDIDATES .. MAX_NUMBER_OF_CANDIDATES

axm19 : $\forall b1 \cdot b1 \in Ballots \Rightarrow$
$((inv\_2(b1) = TRUE) \Leftrightarrow$
$getNumberOfPreferences(b1) \in$
$1 \mathinner{..} getNumberOfCandidates(b1))$
inv_2 requirement: the number of preferences is in range: 1 .. numberOfCandidates

axm20 : $\forall b1 \cdot b1 \in Ballots \Rightarrow$
$((inv\_3(b1) = TRUE) \Leftrightarrow$
$(\forall c \cdot c \in 1 \mathinner{..} getNumberOfCandidates(b1) \Rightarrow$
$getPreferenceForCandidate(b1 \mapsto c) \in 0 \mathinner{..} getNumberOfPreferences(b1)))$
inv_3 requirement: only valid preferences in range 0 .. numberOfPreferences are allocated to each candidate

axm21 : $\forall b1 \cdot b1 \in Ballots \Rightarrow$
$((inv\_4(b1) = TRUE) \Leftrightarrow$
$(\forall p \cdot p \in 1 \mathinner{..} getNumberOfPreferences(b1) \Rightarrow$
$getCandidateForPreference(b1 \mapsto p) \in 1 \mathinner{..} getNumberOfCandidates(b1)$
$))$
inv_4 requirement: only valid candidates in range 1 .. numberOfCandidates are allocated to each preference

Figure 14: Abstract Ballot Invariant Axioms

The modelling and verification of this requirement is trivial in Event-B. Currently we model this requirement as an invariant on the number of Ballots. The only state that changes during tabulation events is the candidate for which a Ballot is currently assigned. Thus, the number of ballots is sure never to change, as we refine the tabulation machine to provide further implementation details.

Our development approach relies heavily on the coherent integration of the Event-B and Java code: our code is not generated from the specifications, rather the formal Event-B is used to document the code and to oblige the programmers to develop test code for invariant properties and functional requirements.

In the cases where we have proven that our Event-B models are correct this should be documented in the code. However, these proofs correspond only to the verification of design steps. They do not validate the natural language requirements and they do not verify that the Java code is a correct implementation. It would be possible to animate the Event-B models to aid validation, and it may be possible — in the future — to automatically generate Java code corresponding to our OO Event-B; however we chose to complement our formal design process with rigorous unit testing. This helps in both the validation of the natural language requirements and in the verification of the Java implementation.

3.4 Software Engineering and Educational Theory — Génie logiciel et théorie pédagogique

Whilst carrying out our research into teaching formal methods, we noticed some more general results concerning the teaching of software engineering. This resulted in publications that attracted interest from researchers from outside the computer science and software engineering communities.

Pendant l'exécution de notre recherche dans l'enseignement des méthodes formelles, nous avons remarqué des résultats plus généraux concernant l'enseignement du génie logiciel. Ce qui a résulté dans des publications ayant l'intérêt des chercheurs hors des communautés de l'informatique et du génie logiciel.

3.4.1 Software engineering as a model of understanding — Génie logiciel comme modèle de compréhension

In 2005, we reported on the first steps towards constructing a theory that: *software engineering provides a good framework for reasoning about how children and adults learn to solve problems.*[OG05c]. This research arose, quite unexpectedly, out of our analysis of problems with teaching first year programming.

En 2005, nous avons rendu compte sur les premières étapes vers la construction d'une théorie : le génie logiciel fournit un bon cadre de raisonnement sur la façon dont les enfants et les adultes apprennent à résoudre des problèmes [OG05c]. Cette recherche est apparue, de façon assez inattendue, à partir de notre analyse des problèmes sur l'enseignement de la programmation en première année.

It is well accepted within the computer science community that first year students find programming difficult. There is an abundance of papers in the proceedings of SIGCSE[9] and ITiCSE[10] that confirm this. One of the major stumbling blocks for students is the abstraction of the problem to be solved from the exercise description[MAD+01]. In order to try to overcome this difficulty, the we introduced a workshop into the first year programming module[OMG+04, OBD+04]; this involved the students working in groups to solve problems. Each student in the class was assigned to a formal group for an entire semester and each group was assigned a separate workspace.

It is our experience — and anecdotal evidence from lecturers in other universities concur with us — that once a student sits in front of a computer, they feel compelled to be using the keyboard or the mouse, and that they do not take the time to map out a solution to the given problem. Therefore, we deliberately prohibited the use of computers during the workshop. We gave the groups a basic framework (which they could change), to tackle the problem. This framework asked them to consider the problem as presented, clarify the kernel of the problem and the product to be produced and restate the problem in their own words. In addition, they could generate ideas/hypotheses through the use of brainstorming, identify the key issues and constraints, and develop a step-by-step set of instructions to solve the problem. The overall objective with the use of these workshops and peer learning groups was to:

- develop the student's problem solving skills,
- develop the student's critical thinking skills,
- encourage alternate approaches to problem solving through group work, and
- encourage deep learning approaches.

[9] ACM Special Interest Group on Computer Science Education — http://www.sigcse.org/
[10] Innovation and Technology in Computer Science Education — http://www.cs.utexas.edu/users/csed/iticse/

It was our belief that if we could improve the students' abilities in these areas there would be a discernible positive result.

During our research we observed that students were reasoning using advanced software engineering concepts:

- Refinement — of data and processes
- Subclassing — extension and specialisation
- Genericity — universal and constrained
- Re-use — composition and aggregation
- Formal reasoning — preconditions, postconditions and invariants

The main contribution of the paper was the insight that this type of reasoning was natural to all students (not just the computer scientists and software engineers).

3.4.2 Re-Use and Plagiarism — Réutilisation et plagiat

At all stages of education and learning, students' reuse of other peoples' work and ideas is fundamental and it is to be encouraged. What is *not* encouraged is when the work of another person is presented as a student's own. This is plagiarism and must *not* be tolerated.

It is each student's responsibility to ensure that when they include (directly or indirectly) the work of others that this contribution is fully and properly acknowledged. Guidelines on the acknowledgment of the work of others can be found in a text by Gordon Harvey [Har98]. Professional bodies (with publishing houses) also provide generic guidelines on plagiarism — the ACM student magazine *Crossroads* is a good example for students [Jor06]. Universities generally have their own policies (or guidelines) on plagiarism. Individual departments may also provide more specific guidelines and one must be careful that these documents are consistent.

In our experience, however welcome these documents are, they do not cover many of the more difficult, technical issues which arise when the work that is being re-used is software. It is the role of a *code of practice* to try and clarify what is

A toutes les étapes de l'éducation et de l'apprentissage, l'utilisation par les étudiants de travaux et d'idées d'autres personnes est fondamentale et cette action doit être encouragée. En revanche, ce qui n'est pas encouragé c'est lorsque le travail d'une autre personne est présenté comme le propre de l'étudiant. Ceci constitue un plagiat est ne peut être toléré.

C'est la responsabilité de chaque étudiant de s'assurer que lorsqu'ils incluent (directement ou non) le travail de quelqu'un d'autre, il faut que cette contribution soit entièrement et correctement reconnue. On peut trouver des indications sur la façon dont reconnaître le travail des autres dans un texte de Gordon Harvey [Har98]. Des corps professionnels (avec des maisons d'éditions) fournissent aussi des directives génériques sur le plagiat — en est un bon exemple [Jor06]. Les universités généralement ont leurs propres politiques (ou indications) sur le plagiat. Les départements individuels peuvent également fournir des directives plus spécifiques et il faut être attentifs à ce que ces documents soient cohérents.

Dans notre expérience, même si ces documents sont les bienvenus, ils ne couvrent pas beaucoup les problèmes techniques les plus difficiles, qui surviennent quand le travail réutilisé est un logiciel. C'est le rôle d'un code de pratiques d'essayer et de clarifier ce que

meant by plagiarism in this context. In [Gib09b] we developed such a code of practice.

signifie plagiat dans ce contexte. Dans [Gib09b], nous avons développé un tel code de pratiques.

In software engineering, software is usually not built from scratch. Normally, already existing software artifacts (from the set of documents and models that are built during the engineering of a software system - analysis, requirements, validation, design, verification, implementation, tests, maintenance, versioning and tools) are re-used, in a wide range of ways, in the construction of a "new" software system.

Software re-use is one of the least-well understood elements of the software engineering process. It is much more challenging to re-use software artifacts in a controlled, systematic way – the quality of software is compromised if a rigorous engineering approach to re-use is not followed. Clearly, any piece of assessed work incorporating the development of software (including final year projects) can be considered to be of "poor" quality if it has relied upon non-rigorous, ad-hoc re-use of other peoples' software.

We proposed that a code of practice for software re-use offers many advantages to students submitting work for evaluation: makes explicit what constitutes plagiarism with respect to software; provides guidelines which, if followed, should ensure that a student is not wrongly accused of plagiarism; defines structures to help examiners to objectively check for plagiarism in a consistent and fair manner; and improves the quality of the software that students produce.

We acknowledged that any code of practice will obviously restrict the way in which software can be developed. Furthermore, such a code will almost certainly be too restrictive in the sense that there are sure to be specific requirements for some system that would be impossible to meet in the time-frame of a project if the code of practice is enforced, but otherwise could possibly be met. For this reason we proposed that exceptional cases be dealt with by the lecturer or project supervisor. The fundamental requirements for the code of practice are that it is: simple to understand, apply and enforce; consistent with wider plagairism policy; and as fair as possible to all students.

In order to guide the formulation of the code of practice, the paper provides concrete examples of acceptable and unacceptable forms of re-use be examined. These examples are not intended to be complete. The examples were chosen because they represent the most common forms of re-use that we have witnessed in final year projects (both acceptable and unacceptable).

3.4.3 Learning From Mistakes — Apprendre de ses erreurs

Teaching science and engineering involves students being asked to solve problems. The two most common approaches are complementary. Firstly, traditional lecturing initially presents the fundamental material that the students need to solve the chosen problem; then students learn about applying this knowledge through problem interaction. Secondly, problem-based learning (PBL) initially engages the students in solving the chosen problem; then the students — often through the guidance of the lecturer — discover

L'enseignement de la science et de l'ingénierie implique qu'on demande aux étudiants de résoudre des problèmes. Les deux approches les plus communes sont complémentaires. Premièrement, l'enseignement traditionnel présente initialement le matériel fondamental dont les étudiants ont besoin pour résoudre le problème choisi ; ensuite les étudiants apprennent comment appliquer cette connaissance au travers d'une interaction avec le problème. Deuxièmement, "problem-based learning" (PBL) engage initialement les étudiants à résoudre le problème choisi ; alors les étudiants - sou-

the fundamental material "themselves". In both approaches, lecturers must identify learning objectives, and ensure that the problem both facilitates students in meeting these objectives and helps to identify which objectives have not been met by particular students. We report, in [Gib09a], on a teaching method which addresses the issue of learning from mistakes in the problem solving process.

vent au travers des conseils de l'enseignant - découvrent le matériel fondamental "eux-mêmes". Dans les deux approches, les enseignants doivent identifier les objectifs d'apprentissage et s'assurer à la fois que le problème facilite les étudiants à remplir ces objectifs et les aide à identifier quels objectifs n'ont pas été remplis par certains. Nous faisons le rapport, dans [Gib09a], sur la méthode d'enseignement qui traite du problème d'apprendre à partir des erreurs dans le processus de résolution d'un problème.

We have observed that students also learn by watching others trying to solve problems. For example, with traditional lectures the lecturer often presents a solution to a classic problem in a manner that simulates, quite artificially, the way in which the problem can be solved. The advantage of this is that the lecturer has complete control over the material being presented. The disadvantage is that the students do not observe a real problem-solving process. In contrast, students watching other students problem solving (during PBL) offers the advantage of them observing a real problem-solving process. However, the lecturer has less control over them meeting their learning objectives.

Balancing the PBL approach with traditional lectures is very difficult. It is very difficult to teach all material using only PBL. One reverts to traditional lecturing when a problem has not been successful in helping students to meet specific learning objectives, or where you have yet to find a problem that has been able to do so. PBL quite often leads, in my experience, to weaker team members being dominated by stronger team members, resulting in the weaker students being less involved and losing motivation. Further, the freedom offered to students in PBL can lead to a chaotic/unstructured learning environment with the lecturer being detached from the process.

In [Gib09a], we proposed a teaching method which addresses this issue: students formulate the problem to be solved and then observe the lecturer trying to solve it. The lecturer guides the student in problem creation and selection, and so ensures that the problem is suitable for meeting the learning objectives. Our experience shows that students learn more from watching the lecturer struggling (and often failing) to solve the chosen problem than from observing the lecturer presenting an obvious solution.

This approach is currently being adopted in other universities, and in disciplines other than computer science and software engineering.

3.4.4 Important Technical Contribution: Refinement is fundamental in learning

The question of *how we learn* continues to be fundamental to teaching in all disciplines and in all age groups. Further, learning models inspire (and are inspired by) understanding of how the brain functions and the concept of intelligence. The notion that refinement is important, first raised by our research in 2005[GO05], has been widely influential in the computer science education community, and beyond. In this subsection we provide some additional technical details on this work (for the interested reader). The original paper — *Software engineering as a model of understanding for learning and problem solving*[GO05] — provides a more complete report.

Searching is a classic problem in learning and in computer science. We report on a multiphase approach to observing children learning about searching; and focus on the application of foundational software engineering concepts (including refinement).

Phase 1: observing pre-requisite understanding

When searching for an object from a collection of objects, we require only that a child can tell when 2 objects are the same. Through experience, we adopt a technique where "sameness" is based on some concrete property of the objects in question (size, shape, colour, texture, etc...). In searching, we have had most success with lengths (the property) of pieces of string (the objects).

First we generate a reasonable number (5 to 20, depending on the age of the children) of pairs of pieces of string; where: the pairs are of equal length, and all pairs have different lengths. For example, a typical problem will have the following pairs, in no particular order, where the numbers represent lengths:

$$\{(4,4);(2,2);(5,5);(10,10);(16,16)\}.$$

Second, we separate the pairs into 2 collections:

$$\{4,2,5,10,16\}\text{and}\{4,2,5,10,16\}$$

Thirdly, we randomly mix each of them; giving, for example:

$$\{10,16,5,4,2\}\text{and}\{4,5,2,16,10\}$$

Finally, we ask the students to put them back into a collection of pairs. (This is similar to the problem of pairing socks, which many of them will be familiar with.) In the process of pairing, we confirm that all the children are able to compare the lengths of pieces of string and match those of equal length. The children do not need to actually solve the pairing problem in order to progress. Now that we know that children know how to check if 2 pieces of string have the same length, we can proceed to searching.In this case, we require only that a child can match a single piece of string with another piece of string in a collection. How you present the collection, and how you constrain their manipulation of the collection is key to observing the learning process.

We demonstrate that we can hide a piece of string in a box, and place a number of pieces of string in a number of boxes (one per box). Finally, we hand them a piece of string and ask them to find the matching string in one of the boxes. However, they are told that they can open only one box at a time; and that when a box is closed it must contain the piece of string that was in it originally. (With younger children it often takes a few minutes for them to understand the "rules of the game".)

Through experience, children are much more enthusiastic and are more likely to actively participate in the sessions when there is an element of competition. In this case, we play the children against each other, playing alternate moves of the game. In this game, a move is looking in a box for the matching string. The first player to match the string wins the game. (Note that this does not require the younger students to be able to count.) All other children act as spectators of each game; and observing the spectators is as insightful as observing the players.

Phase 2: first observations (process refinement)

We first observe the children selecting boxes in a purely haphazard, random manner. Although this, to begin with, is a game of chance, the children still seem to think that, for individual games, the winners are better players than the losers. The first interesting observation is when children realise that they have a better chance of winning if they never look in a box that they have already looked in. This observation usually arises from one (or more) of the children spectators shouting out that a player has already looked in a particular box and that they should choose

another. In terms of software engineering, the children have quite naturally identified and communicated a process refinement.

At this point, we ask children to play against each other using the new, improved approach. However, we preclude the spectators from speaking during a game. Very quickly, it is observed that some of the children have problems remembering in which boxes they have already looked (rather than having problems in following the new better search algorithm). We confirm this by increasing the number of boxes (for younger children only 10, or so, are required.)

Phase 3: second observations (data refinement)

In the searching example, we have observed 3 types of data refinement which they introduced as a specific way of overcoming the problem of having to remember which boxes had already been examined:

- Children searched the boxes in an ordered fashion (left to right, e.g.). Through ordering the boxes in this way, they had only to remember the last box searched in order to perfectly partition the collection of boxes into those already searched and those not yet examined.
- Children marked the boxes already searched (using a pencil, e.g.). Now the partition was explicitly defined by effectively each box having an associated boolean value (marked or not marked).
- Children moved the boxes from a "not yet examined pile" to an "already examined pile".

Phase 4: ordering the data elements (constraining the problem in order to help find a better solution)

The next phase is to order the boxes based on the length of the strings within, before we ask the children to play the game. Very quickly it is observed that not all the children realise that the strings in the boxes are ordered by length. The children who realise the data are ordered are observed playing in a more structured manner. (Note that the notion of ordering in this case introduces a new pre-requisite: it is no longer sufficient for children to be able to check if two strings are of the same length, they now have to be able to tell the relative ordering of the strings.)

Over a period of time, we observe that the children effectively refine their solution[11] to a binary search where they do not always optimise the search by cutting the search space perfectly in two every time they make a guess. They know they need to look to the left or right of the current string box, based on the relative sizes of the search string and the string in the box.

We observe at least one of the children adopting a "nearly perfect" binary search. They are observed trying to explain their *algorithm* to their colleagues. This form of refinement is fundamental to software engineering, is difficult for university students to understand[GM98] and apply, yet is observed in school children when they learn to play a game through competition.

The children are asked if it is possible to play better? Often they make quite solid arguments as to why their solution is optimal. At this stage, we play against them and we always find the string that is being looked for in the first guess. We are employing a perfect hashing function, based on knowledge of additional structure to the data values, to map the string length directly to a particular box. They often accuse us of cheating; and only the more advanced students realise the trick. (We do not explain it to them if they do not see it themselves; and often we are contacted by teachers and parents asking for us to explain the trick!)

[11] We argue that this solution implies an implicit understanding of an algorithm.

Phase 5: generalising the problem

After the children have managed to refine an algorithm for searching for a string of a certain length, we replace the strings by some other physical entities. In most instances, we use weights or balls (of different size). *All* children manage to generalise their solution for strings to other physical entities which can be ordered in some intuitive fashion. This is an example of constrained genericity.

Phase 6: working with abstractions

I tell them that I am thinking of a number between two integer values (usually 0 and 100). The game is that they have to guess the number I am thinking of, by asking me questions to which I am allowed to answer only *yes* or *no*. Quite quickly, we observe that some of the children employ the same algorithm for the "guess the number game" as they do for the "find the string game". We observe that others do not.

Phase 7: observing compositional re-use and subclassing (specialisation)

I tell the children that I am thinking of 2 numbers that add up to 100[12]. I ask them to find both, following the same guessing framework as they have already seen with the strings and "guess the number" games. Before we start to play some games, I ask them to tell me whether this game is more/less difficult than the first (in terms of the number of moves that it will take to find the answer). Surprisingly, most kids say this 2-number game is more difficult because you have 2 numbers to find. In a similar vein, I ask the children to find a number that I am thinking of (between 0 and 100), but I also tell them that it is odd (or even). In this instance, they state that this game is easier than the original (even though the size of the search space is the same.)

I ask the children who think the 2-number game is easier to explain their reasoning. Typically, they provide a constructive specification of how to play the variation using the mechanism that they have refined for playing the original game (the perfect binary search):

> "use the previous search to find the smallest number (which must be in the range 0 - 50), and then calculate the largest number as $(100 - smallest)$."

Of course, this search will, on average, be 1 step quicker that the search for a single number in the range 1 - 100. Some of the children manage to explain this improvement. There are two components that are being re-used here:

- the original binary search, and
- the calculation of the largest (missing number) as a simple subtraction.

A second observation that should be noted is that the children quickly identify a symmetry in the problem that means that they can reformulate their approach in terms of finding the largest (not smallest) element first. This symmetry can be viewed as a form of sub-classing where either approach is functionally correct; but where one implementation may be more or less efficient than another under certain circumstances; in other words, they are each specialisations.

[12] This phase requires us to work with older children who have the ability to count to 100, and perform simple addition and subtraction.

Final Phase: Communication Observation

The number of phases that we execute can vary between 5 and 15 (depending on the school and children). We normally terminate the session with a final phase that helps us to make more meaningful observations with respect to children having really achieved some sort of *algorithmic understanding* of the game they are playing. Where possible, we mix the children from the session with children who have not participated. We routinely observe the children who were involved in the session playing the games against their friends. Some of them keep the algorithmic secrets to themselves in order to improve their chances of winning, others take more delight in explaining the algorithms they have learned to their colleagues.

3.5 Problem Based Learning — Apprentisage par problèmes

3.5.1 Teaching Refinement — Enseigner le raffinement

The Event-B Rodin development environment [ABHV08] is central to our teaching of formal methods. The openness of the platform, combined with our research experience, motivated us to adopt it as early as possible in our teaching. Before Rodin, we had experience of teaching a variety of formal methods; with more recent teaching using B[Abr96] and the Atelier-B tools. Another motivating factor is that our students are all familiar with the Eclipse[dRW04] platform, on which Rodin is built, and this helps them overcome initial feelings of unfamiliarity which often arise from using formal methods tools for the first time.

In [GLR09] we present a Problem-Based Learning (PBL) approach to teaching formal methods, using Event-B and the Rodin development environment. This approach has arisen out of a gradual adoption, over a period of 3 years, of Rodin as the main teaching tool. Just as the concept of refinement is fundamental to what we are trying to teach, we demonstrate that it is also fundamental to the teaching process. Through analysis of a small number of PBL case-studies we argue that the changes to our teaching, supported by Rodin, have started to have a positive impact on our students meeting the specified learning objectives (course requirements). However, we also

L'environnement de développement intégré Rodin (Event-B) [ABHV08] est central pour notre enseignement des méthodes formelles. L'ouverture de la plateforme, combinée avec notre expérience de recherche, nous a motivés à l'adopter aussi tôt que possible dans notre enseignement. Avant Rodin, nous avions l'expérience de l'enseignement d'une variété de méthodes formelles ; avec un enseignement plus récent utilisant B[[Abr96] et les outils de l'Atelier-B. Un autre facteur de motivation est que nos étudiants connaissent tous la plateforme Eclipse [dRW04], sur laquelle Rodin est construit, et ceci les aide à dépasser des sentiments initiaux de mauvaise connaissance qui surviennent souvent lors de l'utilisation des outils de méthodes formelles pour la première fois.

Dans [GLR09], nous présentons une approche PBL pour enseigner les méthodes formelles, en utilisant Event-B et Rodin. Cette approche est apparue de l'adoption graduelle de Rodin comme outil d'enseignement principal sur une période de 3 ans. Comme le concept de raffinement est juste essentiel à ce que nous essayons d'enseigner, nous démontrons qu'il est aussi fondamental au processus d'enseignement. A travers l'analyse d'un petit nombre d'études de cas PBL, nous avançons que les changements pour notre enseignement, supportés par Rodin, ont commencé à avoir un impact positif sur nos étudiants rencontrant les objectifs spécifiques d'apprentissage. Cependant, nous argumentons

argue that much more work needs to be done in order to improve our teaching of formal methods.

également qu'il faut davantage de travail pour améliorer notre enseignement des méthodes formelles.

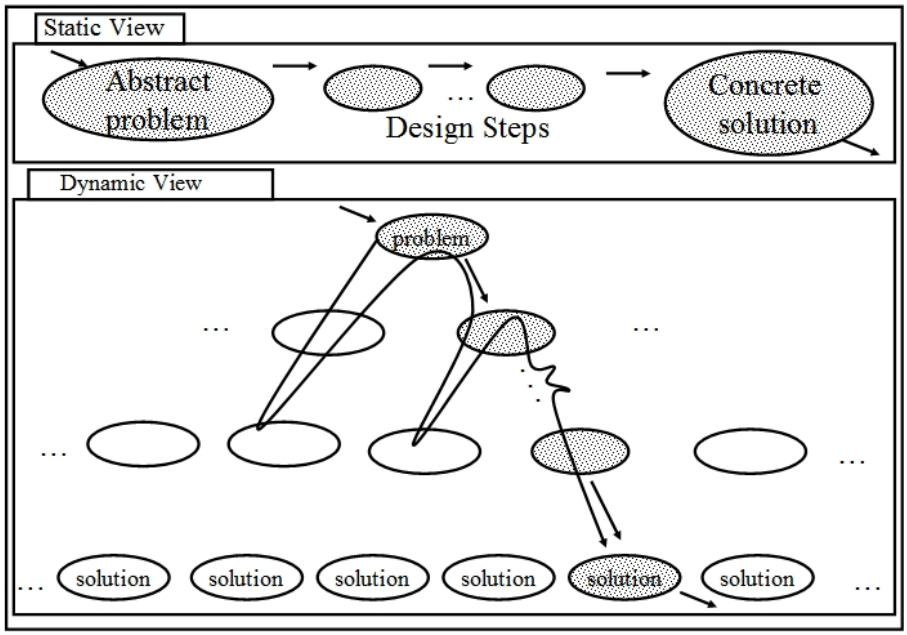

Figure 15: Design as A Dynamic Process: The Learning Objectives

Figure 15 provides a graphical view of our main learning objectives when treating design as a process: (1) to be able to build models at different levels of abstraction, (2) to be able to prove that models at all stages of development are well defined, (3) to be able to validate that models capture precisely the needs of the client, (4) to be able to verify that each design step (from abstract to concrete) is correct, and (5) to be able to manage the process of rolling back a design decision.

We note that these objectives are generic in the sense that the modelling language and modelling tools are unspecified. Our secondary learning objectives are that the students are able to meet the main objectives when modelling with Event-B, and that they are able to use the Rodin tool for (automated) support.

Much like software engineers who refine their models for implementation, formal methods lecturers need to refine their teaching models. When there is a mismatch between what is required and a proposed solution then there are three possibilities: (1) the solution is correct with respect to the requirements specification yet the specification misrepresents the requirements, or (2) the specification correctly represents the requirements but the solution is not correct with respect to the specification, or (3) a combination of the two previous possibilities. In general, when a solution is acceptable it is because the initial requirements were correctly specified and the implementation was correct with respect to these requirements. (In theory, it may be possible that the solution meets the requirements despite the fact that the specification is incorrect. However, this situation is not desirable even though the client may be happy in the short term!)

There are several options when a problem is not meeting a specific learning objective: (1) Replace the problem

with something completely different. (2) Fix the problem by making minor changes. (3) Change the learning objective.

This feedback into our teaching is critical in PBL but it is problematic because: (1) The frequency of change is usually tied to the academic calendar. (2) The mapping relation between learning objectives and problems is not (usually) a bijection, though it should be a total surjection. (3) Analysis of the effectiveness of problems should be done using more than 1 class of students. (4) Developing new problems is time-consuming. The simplest way to overcome these issues is to share and re-use problems between different lecturers and programmes.

Once a problem has been developed that is deemed to be effective, it is very important that one does not break its effectiveness through making change. Lecturers would greatly appreciate a formal notion of refinement with respect to their teaching material. Thus, they could make (verifiable) changes to existing problems knowing that such changes do not compromise their effectiveness (at meeting the learning objectives). Of course, this is currently beyond the state-of-the-art in educational research! However, as teachers we must aspire to achieving such refinements of our problem designs: it will improve our teaching and reduce our workload.

3.5.2 A first programming problem — Comment apprendre la programmation

Piaget's theories [Pia89] continue to be central to school education and it's curriculum. His theory identifies a final key period in a child's life which concerns children older than 11. He argues that at this stage, and not before, children become capable of full logical and mathematical deduction.

Learning how to program is difficult. A major contribution to this difficulty is that programming relies on an implicit understanding of the concept of an algorithm. It has been suggested (by followers of Piaget — in the domain of child psychology and education) that children younger than eleven are unable to understand algorithms. In 2003, we began to teach Java programming in schools in order to investigate and challenge this commonly held belief that young children are not capable of algorithmic reasoning[Gib03]. The researched showed that children as young as seven can demonstrate algorithmic understanding through writing programs. Key to the success of this research was the choice of the correct type of problem. Nought and crosses was an ideal choice — neither too complex nor too simple, and it incorporated all the necessary elements to demon-

Les théories de Piaget [Pia89] restent centrales dans la pédagogie et les programmes scolaires. Sa théorie identifie une période clé finale dans la vie de l'enfant, pour des enfants de plus de 11 ans. Il avance qu'à ce stade, et pas avant, les enfants deviennent plus capables d'une entière déduction logique et mathématique.

Apprendre à programmer est difficile. Une contribution majeure à cette difficulté est que la programmation repose sur une compréhension implicite du concept d'algorithme. Des adeptes de Piaget ont suggéré (dans le domaine de la psychologie de l'enfant et de l'éducation) que les enfants plus jeunes que 11 ans sont incapables de comprendre les algorithmes. En 2003, nous avons commencé à enseigner la programmation Java dans les écoles afin d'enquêter et de défier cette croyance répandue que les jeunes enfants ne sont pas capables de raisonnement algorithmique [Gib03]. La recherche a montré que des enfants aussi jeunes que 7 ans peuvent montrer une compréhension algorithmique en écrivant des programmes. La clé du succès de cette recherche a été dans le choix d'un type correct de problème. Les nœuds et les croix étaient un choix idéal — ni trop complexe, ni trop simple, et il incorporait tous les éléments nécessaires pour démontrer des capacités

strate fundamental programming skills.

de programmation fondamentales.

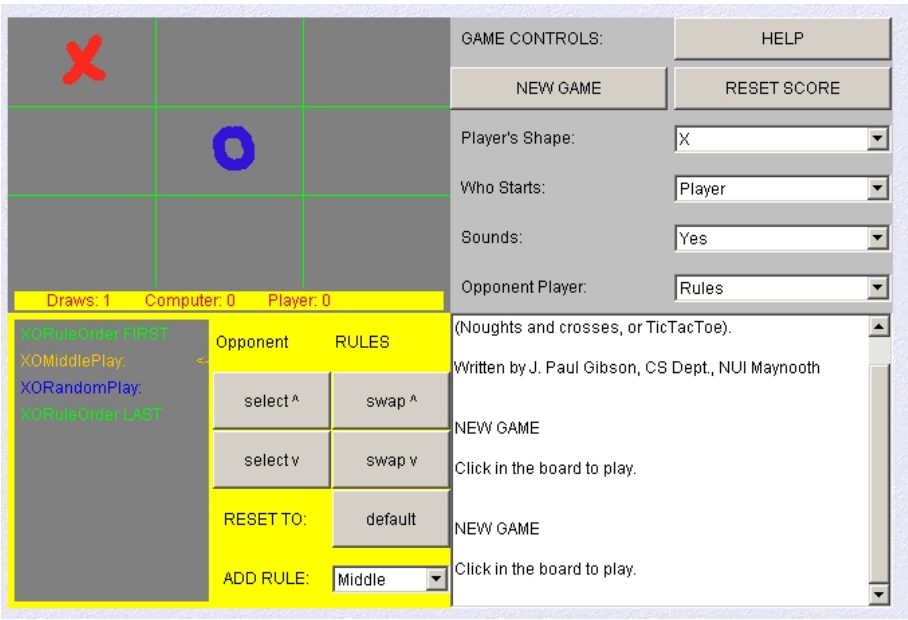

Figure 16: Noughts and Crosses Educational Applet

The teaching approach was to use an interactive Java applet which allowed very young students to construct an intelligent noughts and crosses player through the use of rules. Then, when they had mastered the rule-based approach to programming we showed how each rule could be implemented imperatively. Finally, we encouraged the children to implement, in Java, their own players.

Figure 16 illustrates the Java application that was used to help them to learn to program. Although Java applets had already been successfully deployed as teaching tools in all areas of children's education — Kahn[Kah98] comments on the use of this and other technology with particular emphasis on the interactivity, fun and graphics. Our approach was novel in the way that the applet allows the children to incorporate their programs into the code; and allows them to see how they go from their high-level design — based on sequences of declarative rules — to Java code. One could claim that they were refining their models!

This noughts and crosses problem has been reused in many other computer science and software engineering modules at University. It has been particularly successful in teaching algorithms and data structures and in making students thinking about the importance of invariant properties.

3.5.3 Does PBL work?
— Le APP fonctionne-t-il ?

In 2005, as much of our department's teaching was starting to incorporate problem based learning, we wished to provide some empirical evi-

En 2005, comme beaucoup de l'enseignement de mon département commençait à intégrer "problem based learning", nous souhaitions fournir des preuves

dence of the impact that PBL was having on our teaching. Two different approaches to analysing the impact of PBL on our teaching were investigated:

1. A structured approach where data collection and analysis was well-informed by standard practice in PBL
2. An ad-hoc approach where analysis was primarily subjective based on the experience of the lecturer

We published our research into these alternative approaches in [OG05b]. We concluded that both approaches require a complementary mix of objective and subjective analysis. There is little advantage to be gained, in the short term, from the more structured approach. However, an ad-hoc approach will not scale to reasonable analysis over a number of years of PBL teaching.

empiriques sur l'impact que APP avait sur notre enseignement. Deux différentes approches pour analyser l'impact de l'APP sur notre enseignement ont été examinées :

1. Une approche structurée où la collection et l'analyse de données étaient bien renseignées par des pratiques standards en APP.
2. Une approche ad-hoc où l'analyse essentiellement subjective était basée sur l'expérience de l'enseignant.

Nous avons publié cette recherche avec les deux approches dans [OG05b]. Nous avons conclu que les deux approches requièrent un mélange complémentaire d'analyse subjective et objective. Il y a peu d'avantage à gagner, à court terme, de l'approche la plus structurée. En revanche, une approche ad-hoc ne fonctionnera jamais pour une analyse à travers un certain nombre d'années d'enseignement à l'APP.

3.5.4 Good Problems Are Open Problems — Les bons problèmes sont des problèmes ouverts

The fundamental principle behind Problem-based Learning (PBL) is that the problem is the driving force that initiates the learning. In order to function effectively in a PBL environment a *good* set of problems is required. Solving problems is a vital element within Computer Science and yet the discipline had been slow — in 2005 — to embrace PBL as an approach to learning. The net result meant that there are few *good* PBL problems available to assist new practitioners with implementation. PBL emphasizes a real-world approach to learning, and we presented a RoboCode Competition as a candidate for a *good*, realistic PBL problem within the computer science discipline[OG06]. We list and identify the criteria that categorise a PBL problem as *good* and validate the RoboCode domain against these criteria. We argue that the concept of *freedom* — in different guises — plays a key role in making PBL a *good* mechanism for teaching programming, and

Le principe fondamental derrière l'APP, c'est que le problème demeure le moteur qui amorce l'apprentissage. Afin de fonctionner efficacement dans un environnement APP, un bon ensemble de problèmes est requis. Résoudre des problèmes est un élément vital en informatique et, pourtant, la discipline avait été lente, en 2005, pour adopter l'APP comme une approche à l'apprentissage. Le résultat net signifie qu'il existe peu de bons problèmes disponibles pour assister de nouveaux praticiens dans l'implémentation. L'APP accentue une approche réaliste à l'apprentissage, et nous présentons une "RoboCode Competition" comme bon problème réaliste au sein de la discipline informatique [OG06]. Nous listons et identifions les critères qui catégorisent un problème PBL comme bon et nous validons le domaine "RoboCode" contre ces critères. Nous argumentons que le concept de liberté — dans différentes formes — joue un rôle clé pour rendre PBL comme un bon mécanisme d'enseignement de la programmation, et pour rendre RoboCode comme un bon domaine pour

for making RoboCode a *good* domain for PBL. PBL.

It is our view that in tertiary education students are encouraged to think for themselves, express their views, question, discuss and be free to disagree in their pursuit of knowledge. However, our methods of instruction span a continuum from the prescriptive to the chaotic, with both extremes working against the student. When we are prescriptive we leave no room for the student to seek knowledge, we see him as the empty vessel — or the blank slate — and our job is to fill the vessel or to write on the slate. At the other extreme we provide little or no structure to guide the student in developing their knowledge. A number of positions exist along this continuum where we can provide scaffolding for the students and still allow them the freedom to seek knowledge. The old adage that one learns from one's mistakes allows for creativity, experimentation, and reinforces the belief that making a mistake is not *always* wrong.

It is our experience that it is very difficult to find a problem that is *just stable enough* in the sense that — for a wide range of students — the balance between being non-prescriptive and open, but not so open as to be chaotic, is assured. We argue that the RoboCode problem fulfils this stability criteria.

Software engineering is not constrained by the physical laws and engineering principles that govern other engineering disciples. Within this unconstrained environment, software engineers manage the complexity of choice through consistent application of fundamental conceptual tools such as abstraction; this allows them to continually improve the software engineering process — "re-writing the rules" for development, making the process as much an art as a science. However, a price must be paid for this freedom, which is why design and quality management are such important issues for software engineering. The true role of design is to create a workable solution to an ill-defined problem, thus software design is a creative process that cannot be reduced to a routine procedure, it involves discovery and invention, and it frequently requires intuitive leaps between abstraction levels[Hum05].

The majority of students entering CS1 (in Ireland) believe that because "learning by rote" was successful for them in the past, it will continue to be successful. However, learning to program is not suited to rote learning: it requires the student to understand the problem, develop a solution, implement this solution in a programming language, compile the program, develop test data, test the program, and iterate the debug, compile, test cycle until they have a working solution. Researchers have found that a student's learning style can affect their performance in introductory computer science courses[CS05]. In order to facilitate these diverse learning styles we need to create an environment that allows students the freedom to divest themselves of the necessity they feel to rote learn and to embrace the explorative, creative process of solving problems. They must learn that experimentation is fundamental to all scientific disciplines, and they must become proficient in their use of a computer as their laboratory for designing and implementing computer science experiments.

Competition is an everyday occurrence in the real world and effective problems in PBL emphasize this real world aspect. The RoboCode problem, which we analysed, combines elements of fun, programming, games, AI and competition. It encourages the fun element of creative ideas within the constraints of the RoboCode environment with the challenges of refining these ideas into a workable solution. We would argue that this problem transforms fragile knowledge into a concrete transferable skill that can be applied in new situations. Students develop skills for each stage of the software development process: requirements analysis, design, implementation, and testing; and they can think critically, reflect on their work, conduct tradeoffs and make informed decisions. Our experience shows us that in order for students to gain the maximum benefit from this problem they should have prior experience of working in a team environment. Within PBL the focus is shifted from teaching to learning and this shift in conjunction with a good problem (RoboCode) provides each student with the freedom to think for

themselves, activate their prior knowledge and acquire new knowledge in an explorative and creative way.

Since this paper was published, the Robocode environment has been used in teaching all aspects of software development — from requirements to design to implementation and testing.

3.5.5 Important Technical Contribution: PBL as a good approach to teaching refinement

A major challenge in applying PBL is in developing a set of problems that help students to meet their learning objectives. In particular, a good problem should assist students in discovering and learning about fundamental concepts, rather than a teacher having to explicitly introduce the concepts, as with a traditional lecture. For many years we have not managed to find a good problem for teaching the concept of refinement. In all cases where we used already existing textbook examples the students failed to discover refinement and a more traditional lecture had to be given. However, the problem of a *Purse* of money finally proved successful — it facilitated the students in discovering the refinement concept themselves.

In this subsection we provide some additional technical details concerning this example *Purse* problem (for the interested reader). The original paper — *Sculpturing Event-B Models with Rodin: Holes and Lumps in Teaching Refinement through Problem-Based Learning*[GLR09] — provides a more complete report.

The following requirements were presented to the students:

- A purse contains coins.
- Coins are positive integers, but not all integers have a corresponding coin.
- We wish to start with an empty purse, containing no coins.
- We allow 3 operations:
    - initialise a purse to being empty (containing no coins),
    - add a coin, and
    - pay a certain (integer) sum by removing the correct number of coins from the purse, i.e by removing coins whose total is equal to the sum requested.

Figure 17 shows a graphical representation of the problem that was presented to the students to complement the textual requirements.

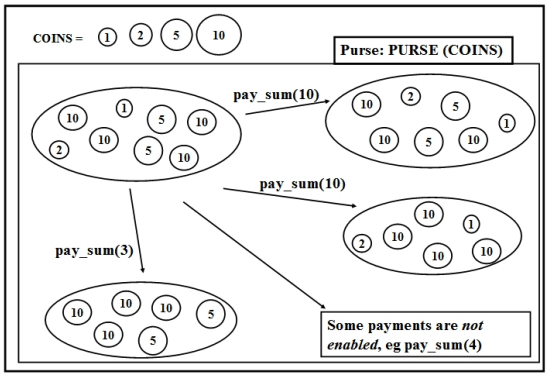

Figure 17: The Purse **pay_sum** Behaviour

It is interesting to note how the students tried to model the `Purse` using Event-B. Firstly, we witnessed the problem of confusing sets with bags as discussed by Habrias[Hab08]. Once students realised that the problem required more than a set of coins (represented as integers) most of them they quickly defined a `Purse` as being a total function from coins to integers. (Some students also chose to specify `Purse` as a partial function from coins to integers, arguing that if a coin was not in the domain then there were no coins of that value in the `Purse`.) The students struggled to specify a generic `Purse`, parameterised by any set of coins. They knew that this type of specification should be possible but had to be shown how to specify this using an abstract `COIN` set.

It was pleasing to see that many of the students then specified the notion of an empty purse in a similar, generic fashion, as shown in figure 18.

```
COINS ⊆ ℕ1
PURSES  = COINS ⟶ ℕ
emptyPURSES ⊆ PURSES
noCOINS ∈ emptyPURSES
emptyPURSES = { z | z∈ PURSES ∧ dom(z) = COINS ∧ ran(z) = {0} }
```

Figure 18: A generic specification of an empty purse

Most students then thought about the operation for paying a certain sum and decided that it was too difficult to specify directly. They were encouraged to think about it in an abstract, nondeterministic, fashion. However, most of them thought that this meant decomposing the payment into component parts. One of the most common ways of doing this was for students to specify the notions of "total" and "remove" (two key terms found in the textual requirements). An example of how a student specified `total` is shown in figure 19.

```
total ∈ PURSES ⟶ ℕ
∀ep· ep∈ emptyPURSES ⇒ total(ep) = 0
∀ p, c,s· p∈ PURSES ∧ c ∈ COINS ∧ s∈ ℕ ∧ c↦s∈ p ⇒ total(p) = c∗s + (total (p ∖ {c↦ 0}))
```

Figure 19: The introduction of a function to calculate the total

At this stage, the lecturer pointed out that the Rodin tool was generating proof obligations with regard to the well-definedness of their `total` specifications, as shown in figure 20.

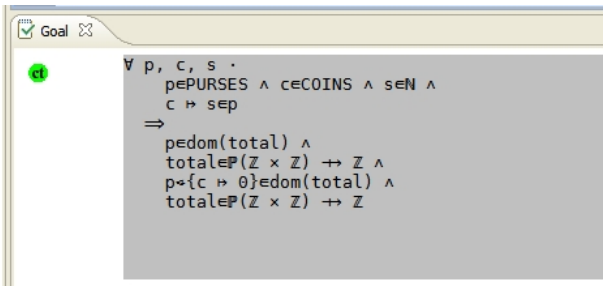

Figure 20: The proof obligation generated for the specification of total

The students experimented with the Rodin tool in order to see which proofs were discharged automatically and which required interaction. Although they did not know how the prover worked (and had received no lectures on the subject) they were able to carry out some proofs simply by "randomly" clicking and instantiating.

By encouraging students to examine different specifications of `total` it often arises that students ask how they can test that their specifications "really work". In essence, they are asking how they can validate that the meaning of `total` corresponds to the requirements. At this stage the lecturer suggests that they formulate simple use cases. The students are able to express the fact that they want to test, for example:

- the total of any empty purse must be 0
- the total of a purse containing two 1c coins should be 2

It was surprising (to us) that, in general, students manage to express these as theorems only after receiving help from the lecturer, as in figure 21.

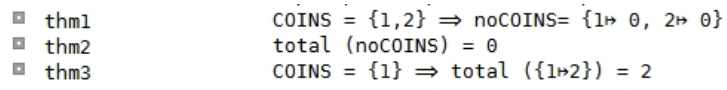

Figure 21: Using theorems to validate understanding and specification

In the process of specifying the `Purse` behaviour we noted that the first design step — of pairing a machine with a context — led to some interesting design decisions. For example, we saw two different specification styles — see figure 22 — for events that update the state of the purse, by adding and removing coins.

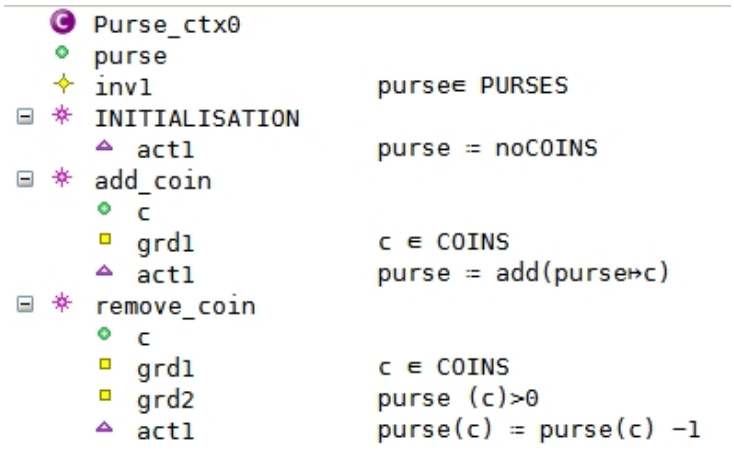

Figure 22: Adding and Removing a coin from a Purse

We note that the `add_coin` event uses an `add` function that has been specified in the context `Purse_ctx0` (see figure 23).

Without going into details, this approach requires additional work when proving the correctness of the context, but leads to a simple proof that the invariant is respected by the `add_coin` event (in the machine). Contrastingly,

```
add∈ PURSES × COINS ⟶ PURSES
∀ p1, p2, c· p1∈ PURSES ∧ p2∈ PURSES ∧ c ∈ COINS ∧ ((p1↦ c)↦ p2) ∈ add ⇒  p2(c) = p1(c)+1
```

Figure 23: The add function defined in the Purse context

the `remove_coin` event's action is specified directly (without an additional "worker" function from the context). This approach means that the proof that `remove_coin` respects the invariant cannot re-use any properties of `remove` that could have been specified in the context.

It was at this stage that the best students normally "discover" refinement. They realize that the `pay_sum` function is highly nondeterministic in the abstract *Purse* machine. Typically, they ask how they can possibly implement this function. Rather than explicitly introducing refinement in Event-B, our next step is to ask the students to implement this function in the programming language of their choice. With a large group of students we expect to see a range of different solutions. The most "nondeterministic" approach is to randomly select subsets of coins until one is found that meets the total requirement. The most "determinisitic" approaches explicitly encode a search algorithm which is guaranteed to search the whole space of possible answers. Again, for a large number of solutions, we can examine which implementations are more efficient than others.

Students are then asked to classify the solutions. Many organise them into a tree where the parent-child relationship is *refinement-like*. The next step is to return to the Event-B models. The teacher then shows them how the relationship that they discovered themselves is actually a fundamental concept in the Event-B language.

This process has proven itself over a number of years with a number of different student groups. The key — we believe — is for the students to discover refinement concept in the (programming) language of their choice; and then to formalise the concept (using a formal language, like Event-B).

3.6 Evaluation and Assessment — Evaluation et contrôle

3.6.1 Cognitive Models of Programing — Modèles cognitifs de programmation

Programming assessment often confuses students. This enforces their belief that programming is difficult, and worsens the problem that lecturers are unsure how best to present the material to aid learning. There is a lack of a clear formal cognitive model of learning programming, and as a result of this, it is difficult for lecturers to find a panacea for teaching programming. There is no optimal path through learning a programming language that guarantees all subjects will be covered in a meaningful manner. One of our goals is to show that each student has their own "optimized" path of learning, which requires a more flexible

Les évaluations de programmation troublent souvent les étudiants. Elles renforcent leur croyance que la programmation est une exercise difficile et cela fait empirer un problème — les enseignants ne sont pas sûrs de quelle est la meilleure façon de présenter le matériel pour aider à l'apprentissage. Il y a un manque de modèle formel cognitif clair pour apprendre à programmer, et ce qui en résulte, c'est qu'il est difficile pour les enseignants de trouver une panacée pour enseigner la programmation. Il n'y a aucun chemin optimal à travers l'apprentissage qui garantie que tous les sujets seront couverts d'une manière significative. Un de nos objectifs est de montrer que chaque étudiant possède son propre chemin "optimisé" d'apprentissage, ce qui requiert qu'un environ-

learning environment tailored, as best as possible, to each student's needs.

Our research paper[TG04b] outlines how the development of a cognitive model will be attempted through the use of student profiling techniques combined with results obtained in an adaptive learning environment. It is believed that if the learning patterns of students can be observed through such a method, useful conclusions about the learning patterns of students can be obtained; e.g., what concepts should be taught together, what are the pre-requisites before certain topics can be covered sufficiently. The paper also discusses what is required to attempt to develop and understand such a cognitive model, and the development methodology that should be employed.

nement d'apprentissage plus flexible soit taillé, aussi bien que possible, pour les besoins de chaque étudiant.

Notre papier de recherche [TG04b] donne un aperçu de comment le développement d'un modèle cognitif est tenté à travers l'utilisation de techniques de profilage de l'étudiant combinée avec les résultats obtenus dans un environnement d'apprentissage flexible. On croit que si les modèles d'apprentissage des étudiants peuvent être observés à travers une telle méthode, des conclusions utiles sur les modèles d'apprentissage des étudiants peuvent être obtenus ; comme par exemple, quels concepts devraient être enseignés ensemble, quels sont les prérequis de valeur avant que certains sujets puissent être suffisamment couverts. L'essai discute également de ce qui est requis pour tenter de développer et de comprendre un tel modèle cognitif, et la méthodologie de développement qui devrait être employée.

To develop a cognitive model of how students learn programming, we must examine the manner in which they learn, which in itself requires understanding of problem solving and software engineering. This was achieved through asking a sample of first year students to undertake several tutorial tests in an adaptive learning environment. First year students were chosen as it is in the earliest stages of learning that students will exhibit the learning patterns that hoped to study. Profiles were developed for each student representing their knowledge in particular areas, as interpreted by the system. The student profiling in this system was performed by processing the students' history of tutorials taken and performance in tests to estimate their ability at programming. This estimation was given assigned a value, so that the students' perceived ability could be monitored and their improvements noted.

The most abstract student profile consisted of a single binary value representing whether the student had sufficient programming ability. A refinement/extension of this simple profile wase a set of boolean values corresponding to whether the student had sufficient programming ability in various programming concepts. This simple profile model however does not carry sufficient information regarding a students' actual ability within the continuum of learning (it is, of course, possible to map a continuum of integer values onto a set of binary values, but this would be an impractical and naive approach to building such a profile model, where the model structure should give some insight into the structure of the learning process itself). Similarly, if a profile was chosen to provide the maximum information, it would carry the entire students' record in an uncompressed form, which would be computationally infeasible to use as a profiling mechanism. This trade-off between computational performance and semantic depth is a major contribution of our research in this area.

The next step in this research was to validate the theory. In [TG04a] we asked whether such Adaptive Learning Environments (ALEs) — based on profiling — could become one of the most useful teaching aids in programming and software engineering courses. The paper concluded that the need for a formal justified cognitive model of programming is clear and its benefits obvious. However, only when we fully understand the process of learning to program, can we teach it properly. Only when we as educators can teach the subject properly, can we consider employing Adaptive Learning Environments to do likewise.

3.6.2 Automated Assessment — Contrôle automatisée

In 2005, based on our previous work on cognitive models, we believed that we could use our more formal understanding of software engineering and problem solving to help in the assessment of students. The goal was to develop automated assessment techniques which were as least as good as the lecturing staff. Such automated assessment could, and should, be applied in all areas of software engineering where formal models (with well-defined syntax and semantics were defined). As a proof-of-concept, we carried out research into this problem with respect to learning how to program[TG05].

The second part of this research was to develop an assessment system and to test its use with students in a real learning environment[TBG06]. The research showed that the automated assessment had a high correlation with the traditional teacher-generated assessment for the top half of the class. For the weaker students, the correlation was weak. Analysis showed that this was due to the weaker students rote-learning, without any clear understanding. This technique led to pass marks in traditional exams but led to failure in our automated assessment.

En 2005, basé sur le présent travail sur les modèles cognitifs, nous avons cru que nous pourrions utiliser notre compréhension plus formelle du génie logiciel et de la résolution de problème pour aider dans l'évaluation des étudiants. Le but était de développer des techniques d'évaluation automatisées qui étaient au moins aussi bonnes que l'équipe enseignante. Une telle évaluation automatisée pourrait, et devrait, être appliquée à tous les secteurs du génie logiciel là où les méthodes formelles (avec une syntaxe bien définie et une sémantique) ont été définies. Comme "proof-of-concept", nous avons entrepris une recherche sur ce problème en concernant l'apprentissage de comment programmer [TG05].

La seconde partie de cette recherche était de développer un système d'évaluation et de tester son utilisation avec les étudiants dans un environnement réel d'apprentissage [TBG06]. La recherche a montré que l'évaluation automatisée était en corrélation élevée avec l'évaluation traditionnelle pour la moitié supérieure de la classe. Pour les étudiants plus faibles, la corrélation était faible. L'analyse a montré que ceci était dû au "rote-learning" des étudiants les plus faibles, sans aucune compréhension claire. Cette technique a conduit à la réussite aux examens traditionnels, mais a conduit à l'échec dans notre évaluation automatisée.

This research described the use of random code generation and mutation as a method for synthesising multiple choice questions which can be used in automated assessment. Whilst using multiple choice questions has proved to be a feasible method of testing if students have suitable knowledge or comprehension of a programming concept, creating suitable multiple choice questions that accurately test the students' knowledge is time intensive. Our paper proposed two methods of generating code which could then be used to closely examine the comprehension ability of students. The first method took as input a suite of template programs, and performed slight mutations on each program and asked students to comprehend the new program. The second method performed traversals on a syntax tree of possible programs, yielding slightly erratic but compilable code, again with behaviour that students can be questioned about. As well as generating code these methods also yield alternative distracting answers to challenge the students. Finally, the paper concluded with a discussion regarding the gradual introduction of these automatically generated questions as an assessment method and discusses the relative merits of each technique.

Since this work has been completed, we have started to use the same techniques for assessment of design (in UML) and specification (in a range of formal models). The research has not progressed as well as for assessment

of beginner programmers; we believe that this is because specification and design problems are much more rich and require a much better formal model of software engineering than we currently have.

3.6.3 Important Technical Contribution: Automated Test Generation

In retrospect, it is surprising how our approach to automatically generating multiple choice questions (MCQs) applied many reasonably advanced software engineering techniques:

- generic programming and mutations for automated test generation,
- ontologies for programming misconceptions, and
- metrics for quality code.

In this subsection we provide some additional technical details concerning this research. The original paper — *Synthesis and Analysis of Automatic Assessment Methods in CS1: Generating intelligent MCQs*[TG05] — provides a more complete report.

To ensure that the MCQs that will be generated by this system will be of a high quality, there are certain requirements that each question must meet:

- **Good quality code** — Any code presented to the students must be of a high quality, and follow all proper guidelines given to students. Obfuscated code and spurious examples within code should not be allowed.
- **No Tricks** — The question should focus on teaching the normal behaviour of programs, not the badly coded exceptions. There should be no tricks in the majority of the questions. (For example in C++ the statement if(i=size)will compile and always returns true, but is bad programming, similarly the expression x+=x++; is valid code but not good practice).
- **Quality Distractors** — The alternative incorrect answers (distractors) must have a suitably high feasibility so as to ensure students are challenged by each question.

These requirements are useful as they serve to both clearly define what can be generated, and also ensure the quality of the output. The approaches to generating both code and questions were influenced by these requirements.

We studied and implemented a variety of different approaches for generating code and questions, ranging from grammatical evolution to various different techniques extracted from genetic programming. Two approaches that we deemed most successful were that of template based mutations, and random walks through a predefined syntax tree. The final implementation — combining both approaches — incorporated sophisticated code quality metrics, ontological models and code mutation.

Experimental analysis demonstrated the utility of the template based approach. Template based mutation uses the mutation phase from genetic programming and applies it to a pre-approved population of problems. In practice this means the lecturer will supply what they consider to be a suitable suite of problems, each one consisting of of a piece of code, and a question regarding its behaviour.

The mutations of the code take the form of minor changes that guarantee different behaviour from the program. The result of this is that one input template can result in a multitude of different questions generated. The mutations are performed by randomly selecting a set of substitutions that should have an effect on the outcome of the code. These substitutions are usually related to the topic under examination and the student misconceptions. For example if a lecturer wished to examine students' knowledge of boolean operators, the substitutions could replace < with <=, = with != etc.... The new program is then compiled and the correct answer is then retrieved through execution.

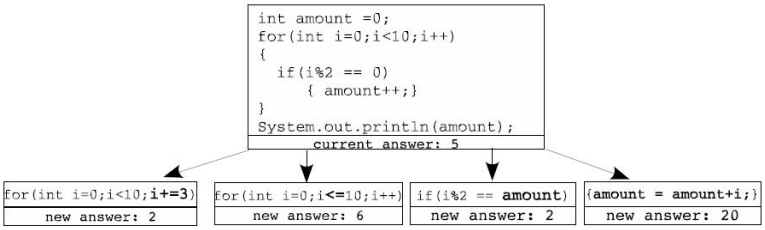

Figure 24: A simple code mutation

Figure 24 gives a short example of how a piece of code can be automatically mutated to produce several different pieces of code, each with alternative outputs. Possible solutions are filtered using code quality metrics.

This study demonstrates that innovative teaching and learning tools often require researchers to master advanced software engineering tools and techniques. In order to implement "intelligent" evaluation systems one must first acquire an excellent understanding of the domain being examined, and build a formal domain model.

Teaching Bibliography

[ABHV08] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, and Laurent Voisin. A roadmap for the Rodin toolset. In *Abstract State Machines, B and Z, First International Conference ABZ 2008*, volume LNCS 5238, pages 347–351, September 2008.

[Abr96] Jean-Raymond Abrial. *The B Book - Assigning Programs to Meanings*. Cambridge Univ. Press, 1996.

[Bab87] R.L. Baber. *The Spine of Software — Designing Provably Correct Software: Theory and Practice, or: A Mathematical Introduction To The Semantics Of Computer Programs*. John Wiley and Sons, 1987.

[Bel00] Tim Bell. A low-cost high-impact computer science show for family audiences. *23rd Australasian Computer Science Conference*, 00:10–16, 2000.

[BG81] Robert Balzer and Neil M. Goldman. Principles of good software specification and their implications for specification languages. In *AFIPS National Computer Conference*, volume 50 of *AFIPS Conference Proceedings*, pages 393–400. AFIPS Press, 1981.

[BJA82] D Bjørner, CB Jones, and D Andrews. *Formal specification and software development*. Prentice Hall, 1982.

[Bol88] T. Bolognesi. Fundamental results in the verification of observational equivalence: a survey. In H. Rudin and West C.H., editors, *Protocol Specification, Testing and Verification VII*. North-Holland, 1988.

[Boo99] Grady Booch. *The UML User Guide*. Addison Wesley, 1999. ISBN 0201571684.

[CB82] D. H. Clements and M. T. Battista. Constructivist learning and teaching. *Arithmetic Teacher*, 38(1):34–35, 1982.

[CGR93] Dan Craigen, Susan Gerhart, and Ted Ralston. An international survey of industrial applications of formal methods. Nistgcr 93/626, U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology, Computer Systems Lab., Gaithersburg, MD 20899, 1993.

[CS05] A.T. Chamillard and Ricky E. Swardl. Learning styles across the curriculum. In *ITiCSE 2005: Proceedings of the 10th Annual SIGCSE Conference Innovation and Technology in Computer Science Education*, pages 241–245, New York, NY, USA, 2005. ACM Press.

[Cus89] E. Cusack. Refinement, conformance and inheritance. In *Open University workshop on the theory and practice of refinement*, 1989.

[DB04] C. Neville Dean and Raymond T. Boute, editors. *Teaching Formal Methods, CoLogNET/FME Symposium, TFM 2004, Ghent, Belgium, November 18-19, 2004, Proceedings*, volume 3294 of *Lecture Notes in Computer Science*. Springer, 2004.

[DeN87] R. DeNicola. Extensional equivalence for transition systems. *Acta Informatica*, 24:211–237, 1987.

[Dil90] A. Diller. *An Introduction To Formal Methods*. John Wiley and Sons, 1990.

[dR04] Sylvia da Rosa. Designing algorithms in high school mathematics. In C. Neville Dean and Raymond T. Boute, editors, *Teaching Formal Methods, CoLogNET/FME Symposium, TFM 2004*, volume 3294 of *Lecture Notes in Computer Science*, pages 17–31, Ghent, Belgium, 2004. Springer.

[dRW04] Jim des Rivières and John Wiegand. Eclipse: A platform for integrating development tools. *IBM Systems Journal*, 43(2):371–383, 2004.

[Gib97] J. Paul Gibson. Feature requirements models: Understanding interactions. In Petre Dini, Raouf Boutaba, and Luigi Logrippo, editors, *Feature Interactions in Telecommunications Networks IV, (FIW 1997)*, pages 46–60, Montréal, Canada, 1997. IOS Press.

[Gib00] J. Paul Gibson. Formal requirements engineering: Learning from the students. In Doug Grant, editor, *12th Australian Software Engineering Conference (ASWEC 2000)*, pages 171–180. IEEE Computer Society, 2000.

[Gib03] J. Paul Gibson. A noughts and crosses Java applet to teach programming to primary school children. In James F. Power and John Waldron, editors, *Proceedings of the 2nd International Symposium on Principles and Practice of Programming in Java (PPPJ 2003)*, volume 42 of *ACM International Conference Proceeding Series*, pages 85–88, Kilkenny City, Ireland, 2003. ACM.

[Gib05] J. Paul Gibson. E-voting requirements modelling: An algebraic specification approach (with cafeobj). Report NUIM-CS-TR-2005-14, Department of Computer Science, National University of Ireland, Maynooth., 2005.

[Gib08a] J. Paul Gibson. Formal methods — never too young to start. In Zoltan Istenes, editor, *Formal Methods in Computer Science Education (FORMED 2008)*, pages 151–160, Budapest, Hungary, March 2008. Accepted for publication in ENTCS.

[Gib08b] J. Paul Gibson. Weaving a formal methods education with problem-based learning. In T. Margaria and B. Steffen, editors, *3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*, volume 17 of *Communications in Computer and Information Science (CCIS)*, pages 460–472, Porto Sani, Greece, October 2008. Springer-Verlag, Berlin Heidelberg.

[Gib09a] J. Paul Gibson. Challenging the lecturer: Learning from the teacher's mistakes. In Fiona O'Riordan, Fergus Toolan, Rosario Hernandez, Robbie Smyth, Brett Becker, Kevin Casey, David Lillis, Geraldine McGing, Majella Mulhall, and Kay O'Sullivan, editors, *ICEP 09 Conference Papers: Engaging Pedagogy*, pages 61–71, Dublin, Ireland, November 2009. Griffith College Dublin.

[Gib09b] J. Paul Gibson. Software reuse and plagiarism: A code of practice. *SIGCSE Bull.*, 41(3):55–59, 2009.

[GLR08] J. Paul Gibson, Eric Lallet, and Jean-Luc Raffy. How do I know if my design is correct? In Zoltan Istenes, editor, *Formal Methods in Computer Science Education (FORMED 2008)*, pages 61–70, Budapest, Hungary, March 2008. Accepted for publication in ENTCS.

[GLR09] J. Paul Gibson, Eric Lallet, and Jean-Luc Raffy. Sculpturing Event-B models with Rodin: "holes and lumps" in teaching refinement through problem-based learning. In *From Research to Teaching Formal Methods - The B Method (TFM B'2009)*, pages 7–21, Nantes, France, 2009. APCB.

[GM98] J. Paul Gibson and Dominique Mery. Teaching formal methods: Lessons to learn. In Sharon Flynn and Andrew Butterfield, editors, *2nd Irish Workshop on Formal Methods (IWFM 1998)*, Electronic Workshops in Computing, Cork, Ireland, July 1998. BCS.

[GO05] J. Paul Gibson and Jackie O'Kelly. Software engineering as a model of understanding for learning and problem solving. In *ICER '05: Proceedings of the 2005 international workshop on Computing education research*, pages 87–97, New York, NY, USA, 2005. ACM.

[Hab08] Henri Habrias. Teaching specifications, hands on. In *Formal Methods in Computer Science Education (FORMED)*, pages 5–15, March 2008.

[Har98] Gordon Harvey. *Writing with sources: A guide for students*. Hackett Publishing Company, 1998.

[Hum05] Watts S. Humphrey. *PSP: A Self-Improvement Process for Software Engineers*. Pearson Education, Inc., NJ, 2005.

[Jor06] Chris Jordan. At a crossroads: plagiarism. *Crossroads*, 13(1):2–2, 2006.

[Kah98] Ken Kahn. *The Design of Children's Technology*, chapter Helping children to learn hard things: Computer programming with familiar objects and actions. Morgan Kaufmann, 1998.

[Lý94] Gérard Lévy. *Algorithmique Combinatoire: Méthodes Constructives*. DUNOD, 1994.

[MAD+01] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In *ITiCSE-WGR '01: Working group reports from ITiCSE on Innovation and technology in computer science education*, pages 125–180, New York, NY, USA, 2001. ACM Press.

[Mar02] Robert C. Martin. *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, 2002. ISBN 0135974445.

[OBD+04] J. O'Kelly, S. Bergin, S. Dunne, P. Gaughran, J. Ghent, and A. Mooney. Initial findings on the impact of an alternative approach to problem based learning in computer science. In *Problem-Based Learning International. Conference 2004: Pleasure by Learning*, July 2004.

[OG05a] J. O'Kelly and J. Paul Gibson. PBL: Year one analysis — interpretation and validation. In *PBL In Context — Bridging Work and Education*, 2005.

[OG05b] Jackie O'Kelly and J. Paul Gibson. PBL: Year one analysis — interpretation and validation. In *PBL In Context — Bridging work and Education*, Lahti, Finland, 2005.

[OG05c] Jackie O'Kelly and J. Paul Gibson. Software engineering as a model of understanding for learning and problem solving. In *ICER '05: Proceedings of the 2005 international workshop on Computing education research*, pages 87–97, New York, NY, USA, 2005. ACM.

[OG06] Jackie O'Kelly and J. Paul Gibson. Robocode & problem-based learning: a non-prescriptive approach to teaching programming. *SIGCSE Bull.*, 38(3):217–221, 2006.

[OMG+04] J. O'Kelly, A. Mooney, J. Ghent, P. Gaughran, S. Dunne, and S. Bergin. An overview of the integration of problem based learning into an existing computer science programming module. In *Problem-Based Learning International. Conference 2004: Pleasure by Learning*, July 2004.

[Par98] David Lorge Parnas. Software engineering programmes are not computer science programmes. *Ann. Software Eng.*, 6:19–37, 1998.

[Pia89] Jean Piaget. *The Jean Piaget Bibliography*. Jean Piaget Archives Foundation, 1989. ISBN:288288012X.

[Pol71] G. Polya. *How to Solve It*. Princeton University Press, November 1971.

[Ree92] Jack W. Reeves. What is software design. *C++ Journal*, 2(2), 1992.

[Rob04] Ken Robinson. Embedding formal development in software engineering. In C. Neville Dean and Raymond T. Boute, editors, *Teaching Formal Methods, CoLogNET/FME Symposium*, volume 3294 of *Lecture Notes in Computer Science*, pages 203–213, Ghent, Belgium, 2004. Springer.

[TBG06] Des Traynor, Susan Bergin, and J. Paul Gibson. Automated assessment in CS1. In *ACE '06: Proceedings of the 8th Austalian conference on computing education*, pages 223–228, Darlinghurst, Australia, 2006. Australian Computer Society, Inc.

[TG04a] Des Traynor and J. Paul Gibson. Implementing cognitive modelling in CS education: Aligning theory and practice of learning to program. In Kinshuk, Demetrios G. Sampson, and Pedro T. Isaías, editors, *Cognition and Exploratory Learning in Digital Age CELDA 2004*, pages 533–536, Lisbon, Portugal, 2004. IADIS.

[TG04b] Des Traynor and J. Paul Gibson. Towards the development of a cognitive model of programming: a software engineering proposal. In E. Dunican and T.R.G. Green, editors, *Psychology of Programming Interest Group 16th annual workshop (PPIG 2004)*, pages 79–85, 2004.

[TG05] Des Traynor and J. Paul Gibson. Synthesis and analysis of automatic assessment methods in CS1: generating intelligent MCQs. *SIGCSE Bull.*, 37(1):495–499, 2005.

4 A RESEARCH PROPOSAL: A SPL FOR E-VOTING — UNE PROPOSITION DE RECHERCHE : UNE SPL POUR LE VOTE ÉLECTRONIQUE

In answer to the question of why it happened, I offer the modest proposal that our Universe is simply one of those things which happen from time to time.
Edward P. Tryon

Knowledge is the only instrument of production that is not subject to diminishing returns.
John Maurice Clarke

4.1 Abstract — Résumé

All engineering requires discipline in order to construct solutions to real-world problems; and the foundation of this discipline must be scientific rigour. Science is the synthesis and analysis of mathematical models, based on observation and experiment, used to capture properties, and define abstractions, of the observed world. Thus, modelling and abstraction are fundamental to engineering.

Software engineering can be regarded as the establishment and application of sound principles and methods in the development of software for execution on computers. Computer Science is the study of the mechanical processes that permit the representation, processing, storage and communication of information. Thus, software engineers must work with models and abstractions of computers, computations and information. Software engineering lacks discipline for two complementary reasons. Firstly, there are only a few well-established principles that bridge the gap between computer science and software engineering. Secondly, where such principles exist practising engineers often fail to apply them.

Formal methods incorporate the principles of moving from abstract models of problems to be solved (the *what*) to concrete models of solutions involving computers (the *how*). They are founded

Toute ingénierie requiert de la discipline afin de construire des solutions aux problèmes du monde réel ; et le fondement de cette discipline doit être la rigueur scientifique. La science est la synthèse et l'analyse de modèles mathématiques, basées sur l'observation et sur l'expérience, et utilisées pour capturer les propriétés et pour définir les abstractions du monde observé. Ainsi, la modélisation et l'abstraction sont fondamentales pour l'ingénierie.

Le génie logiciel peut être envisagé comme l'établissement et l'application de principes solides et de méthodes, dans le développement de logiciels pour exécution sur ordinateurs. L'informatique est l'étude de procédés mécaniques qui permettent la représentation, le traitement, le stockage et la communication de l'information. Ainsi, les ingénieurs de logiciels doivent travailler avec modèles et abstractions d'ordinateurs, calculs et information. Le génie logiciel manque de discipline pour deux raisons complémentaires. Premièrement, il n'y a que peu de principes bien établis faisant le lien entre informatique et génie logiciel. Deuxièmement, là où de tels principes existent, les ingénieurs pratiquant manquent souvent de les appliquer.

Les méthodes formelles incorporent les principes du passage de modèles abstraits de problèmes à résoudre (le quoi) vers des modèles concrets de solutions impliquant des ordinateurs (le comment). Elles sont fondées sur des modèles mathématiques de computabilité, complex-

on mathematical models of computability, computational complexity, communication and correctness. Yet, few practising software engineers know when or how to apply these methods. It is therefore not surprising that the quality of software systems is compromised and that such systems often fail to correctly solve the problem for which they were developed.

Electronic voting systems provide excellent examples of poorly engineered software. Analysis of such systems can, and should, help us to identify where engineers failed to apply fundamental (software) engineering principles. In particular, they provide an opportunity to illustrate where the application of formal methods could have helped the developers to have constructed much better quality solutions to the e-voting problem.

In this proposal we analyse e-voting and show that many problems that arose could have been avoided if rigorous software engineering practices, including formal methods, had been applied. We argue that educators, and the education system, are as responsible for poorly built software systems as the (so called) engineers who developed them. Finally, we argue that the next generation of electronic voting systems need to be better engineered than the current generation; and we propose that the most promising approach to achieving this requires the development of a *formal software product line*.

ité, de communication et de justesse. Cependant, peu d'ingénieurs en logiciels qui pratiquent savent quand ou comment appliquer ces méthodes. Il n'est donc pas surprenant que la qualité des systèmes de logiciels soit compromise et que de tels systèmes échouent souvent à résoudre correctement le problème pour lequel ils ont été développés.

Les systèmes de vote électronique fournissent d'excellents exemples de logiciels pauvrement construits. L'analyse de tels systèmes peut, et devrait, aider à identifier là où les ingénieurs ont échoué à appliquer les principes fondamentaux du génie logiciel. Plus spécialement, ils fournissent une bonne illustration de là où l'application des méthodes formelles aurait pu aider les développeurs à construire des solutions de bien meilleure qualité au problème du vote électronique

Dans ce projet de recherche, nous analysons le vote électronique et montrons que de nombreux problèmes rencontrés auraient pu être évités si les pratiques rigoureuses du génie logiciel incluant les méthodes formelles avaient été appliquées. Nous argumentons que les enseignants, ainsi que le système éducatif, sont tout autant responsables de la pauvre construction de systèmes logiciels que les (soi-disant) ingénieurs qui les ont développés. Au final, nous argumentons que la prochaine génération des systèmes de vote électronique nécessite une meilleure construction que la génération actuelle ; et nous suggérons que l'approche la plus prometteuse pour atteindre cela requiert le développement d'une *ligne de produit de logiciels formels*.

4.2 Motivation

The software in e-voting machines has not, in general, been well-engineered[GM08]. Many governments have chosen to adopt e-voting as a show-case for innovative technology[SL03]. It is a poor reflection on the profession of software engineering that the software in these systems is, in general, neither trusted nor trustworthy[RR06]. We propose that the software engineering community should look upon this as an opportunity to demonstrate just how much software engineering methods, techniques and tools have evolved since the turn of the century[PGL00]; and that the software industry is now mature enough to develop e-voting machines that are highly dependent on software and that are highly dependable.

Software Product Lines (SPLs) [CN02] are attracting attention in the area of applied software engineering research. The challenge, which this article addresses, is to demonstrate how and why an e-voting SPL could be built. E-voting systems correspond in terms of size and complexity to those reported in a number of SPL case

studies [Bos99b]. The number of variations across systems[SP06] is large enough to merit an SPL approach, but not so large as to be unmanageable. Furthermore, these systems exhibit a large amount of common functionality and so the potential for re-use is high. The aspect of e-voting that may be more challenging is that the software may be considered (safety or mission) critical [MG03]. However, recent research suggests that SPLs can be used to develop safety critical systems [Liu07].

Many of the problems that have arisen in the domain of e-voting have arisen because of poorly specified requirements and standards documents [MG06]. It has been proposed that a comprehensive domain analysis be carried out before standards are re-engineered [GM08]. The resulting domain models should provide the foundations upon which standards could be built; and they would also play a major role in the development of an e-voting SPL.

We note that analyses of existing systems — such as the state of Ohio's EVEREST report[BEH$^+$08] — have identified verification issues directly related to foundational concepts in software product lines: "parameterized families of components" [McI68], a "family of related programs" [Dij72, Par76], and "structuring commonality and variability according to features"[KCH$^+$90].

4.3 Historical Context

4.3.1 Secret Ballots

One can trace back the genesis of voting machines to the high level requirement that the voting process should ensure, as much as possible, that a voter records a sincere choice through an expression of free will; and, as such, we must strive for a voting process which guarantees that a voter cannot be coerced through intimidation or bribery.

This high level requirement is usually refined into one of secrecy (or anonymity). Informally, there should be *no* link between votes (as recorded on ballots) and voters. Given a set of N votes then the probability that any individual vote belongs to a specific voter is $\frac{1}{N}$, and full disclosure of all voting information, including all votes recorded on ballots, should not change this probability. In particular, there should be no way of deducing that a voter did (probability 1) or did not (probability 0) vote for a particular candidate (or choice)[13].We note that this requirement is not the same as keeping secret whether or not a particular person has actually recorded a vote.

Evidence of secret voting goes as far back as ancient Greece. The right to secret ballots has long been a legal requirement in many countries around the world. For example:

- In France, article 31 of the Constitution of 1795 states that all elections are to be held by secret ballot.
- In Australia, the colonies of Victoria and South Australia enacted legislation for secret ballots in 1856.
- In the UK, the Ballot Act of 1872 requires that British general elections to Parliament and local government election use the secret ballot.
- In Canada, in 1873, the province of British Columbia enacted the countrie's first secret ballot legislation.
- In the United States, most states required secret ballots (refered to as Australian ballots) soon after the presidential election of 1884. However, it took a total of 7 years before all states had outlawed non-secret voting.
- In Argentina, in 1912, the Law of Universal Ballot required secret (compulsory) votes.

[13]We note that such a deduction can be made if $N = 1$, which would happen if there was only a single voter for which we did not know how they had voted.

- In Ireland, in 1937, the Consitution of Ireland[14] makes reference to secret ballots in articles 12 and 16, and secret postal ballots in article 18.

Secret voting is now seen as a property of a mature democratic process. However, as we see in more detail later in this thesis, the notion of secrecy is not well-defined and open to interpretation. For example, in the UK it is possible to link a "secret" ballot to the voter that cast it, through a particular numbering scheme for ballot papers and electors[15]. Another recent example is in Zimbabwe where the "secrecy" of a ballot can be compromised by the marking of votes by the election officials, where a presiding officer's mark may be placed in a position where concealment of the voter's own mark may be compromised. If a voter perceives that the presiding officer could see their vote then this may compromise their free expression.

The Australian ballot system, from 1856, provides a much copied, prescriptive mechanism for implementing secrecy, and preventing anyone from linking a particular voter to a particular ballot. The four key properties are:

- an official ballot (being printed at public expense),
- the ballot contains only the names of the nominated candidates and/or parties and/or proposals,
- the ballots are distributed only at the polling station, and
- the ballots are marked in secret (in a polling booth) at the polling station.

The significance of the *Australian Ballot* in the history of voting should not be underestimated[Bre06]:

> *Voting by ballot, in "secret" . . . was in use in America and Europe well before being implemented in Australia. This was the secret ballot many demanded for Australia, but they got something else: the Australian ballot, wholly original, with identifying features — such as the government printed ballot paper — previously unimagined. The Australian ballot was not the world's first secret ballot; it was much more important than that.*

These secret ballots are specifically designed to prevent anyone from linking voter to ballot. Although not related to the secrecy requirement, they have the additional advantage that they can also be designed in such a way as to eliminate bias in the voting process. The question of bias also arises in electronic voting interface design[BLS+03].

4.3.2 Mechanical (Gear and Lever) Voting Machines

The inventor Thomas Edison took out a patent[16] on a "electrographic vote recorder" in 1869, but his device was never mass-produced for use in elections.

The earliest mass-produced voting machines appeared in the USA in the 1890s. At this time, there was a widely held perception, in the USA, that the Australian ballot system had inherent weaknesses that needed to be addressed. Firstly, there were claims that government-printed ballots were designed to keep illiterate voters away from the polling booth. Secondly, State and local government officials called for investment in voting machines because of the increasing length and complexity of ballots with multiple candidates and referenda, and the growing size of the electorate (doubling, in one instance, with the enfranchisement of women).

[14] Bunreacht Na hÉireann

[15] It is a legal requirement that was introduced in 1983 as a security arrangement so that if there was an allegation of fraud, false ballot papers could be identified. However, the process of matching ballot papers to voters is permissible (and possible) only if an Elections Court requires it, and this procedure is extremely unlikely to be invoked.

[16] This was his first patent out of more than a thousand in all!

Thie first mechanical (gear and lever) voting machine was patented by Alfred J. Gillespie and manufactured by the Standard Voting Machine Company. In order to meet the requirement for secrecy it incorporated a voter-activated mechanism that drew a privacy curtain around the voter as well as unlocking the machine's levers for voting. Gillespie worked for a number of years with inventor Jacob Myers, refining their machine and making public demonstrations (starting in 1892); and in 1898 they founded the Automatic Voting Machine Company. By the 1920s, a significant market for these machines had developed across many of America's most populated areas.

By 1960, machines based on Myers' design recorded and counted the majority of the votes cast in the United States. However, to manage the complexities of American elections, the machines incorporated dozens of intricate interacting sub-systems and over 25,000 moving components. There are a number of lessons that we can take from the engineering of these gear-and-lever machines:

- **Simple, sound election procedures mitigate vulnerabilities —** Lever machines are vulnerable to certain kinds of fraud, but these vulnerabilities, and associated attacks, were well-understood and consequently were easily prevented by sound election procedures.
- **Keep the User Interface Consistent —** From 1898 through the early 1960s, the gear-and-lever voting machine was promoted as an ideal voting technology. Though its internal mechanism changed over the years, the machine's "three steps to vote" never changed:
  - Pull the handle to close the curtains of the polling booth.
  - Turn the voting levers over the names of your chosen candidates to expose the selctions.
  - Pull the handle back to register your vote and to reopen the curtains.
- **Trust and Transparency —** American voters and election officials were quick to trust the gear-and-lever voting machines: there was no need for (or demand for) an independent (paper) audit trail. Only a few users expressed concerns and peer-pressure, arising from pride in the technology, overcame small pockets of distrust. The key property to the accetpance of the technology was that it was transparent. For every action of the voter (user) there was a clear reaction from the machine, as its internal state changed due to the movement of its internal components.
- **Rigorous Validation of Votes —** One of the most innovative aspects of the gear-and-lever technology was its validation of votes: from the earliest versions the machines were configured to prevent overvotes by locking out other candidates when one candidate's switch is flipped.
- **Human Involvement and Common Sense —** For each voter, when they validate their vote, a lever is pulled which increments the appropriate counters. The final counts (for that particular machine) would then be hand written by an election official at the conclusion of voting. The election offical is not expected to recount the vote, but they would be expected to report on any counts that appeared unusual, such as no votes being recorded for a popular candidate. The engineers of that era acknowledged that there was no way to replicate this common sense functionality with their technology and so understood the importance of human involvement.
- **Familiarity —** Instructional models were used to acquaint voters with the operational features of the actual machine. Posters were used to familiarize voters with the gear-and-lever ballot formats.

Although these machines were a remarkable technological achievement they were expensive to make and difficult to store and transport. Thus, there were economic pressures to adopt a more affordable technology, namely punch cards.

4.3.3 Punch Card Voting Machines

In the USA, the first punch-card voting systems appeared in the 1960s. In contrast to gear-and-lever machines, they had significantly reduced production costs and weighed only a few kilograms. As a result, gear-and-lever voting machines became less popular and by the 1980s they were no longer mass produced[17].

The first voting machine to turn a punch card into a ballot was developed by Martin Coyle, after whom it was named, and was first used by voters in 1961. Punch card systems employ cards — where, as with the Australia Ballot system, candidates and referendum issues are printed directly on the card — and voters use a small device for punching holes as a means of recording a choice beside a candidate or issue. After voting, the voter may place the ballot in a ballot box, or the ballot may be fed into a computer vote-tabulating device at the polling station.

There are a number of lessons to be learned from punch card machines:

- **Inherit good requirements from previous systems —** The gear-and-lever machines introduced a rigorous process for validation of votes to ensure that undervoting could not occur on a submitted ballot. This was not an original requirement of the system but it was clearly good engineering practice. The manufacturers of punch card systems developed their machines in order to offer validation functionality: voters would roll their ballot into a machine and a red light would signal an over- or under- vote; whilst green would signal a valid ballot that could be recorded.
- **A voter verifiable audit ("paper trail") —** Voters would remove their ballot from the machine (signalling green) and a counter (of the total number of votes) would be incremented. The ballot would then be counted by hand, or collected for centralised electronic processing. In the case of dispute, the paper ballots could be recounted and the count process could be fully audited by the voters.
- **Humans are not machines —** Machines can consistently punch holes in cards cleanly, but humans cannot and this results in ballots that are open to interpretation, due to hanging chads[18], swinging chads, tri chads, pregnant chads, dimpled chads, etc ...
- **Machines are not humans —** When counting votes recorded on punched cards, a major problem arises when the holes are not precisely aligned with the ballot. Humans have no problems in correctly interpreting the vote when the alignment is reasonably consistent; however, such flexibility is often lacking in punch card counting machines.

In the 2000 US Presidential elections problems (with non-standard chads) in Florida led to the reliability of punch card ballots to be called into question[Cra01]. This resulted in the public questioning the trustworthiness of all voting technologies, and a higher public awareness of problems with election procedures.

4.3.4 Marksense (Optical Scan)

An alternative to punching holes in cards is to require the voter to mark a ballot (paper) in a specific way[19] so that the process of correctly interpreting the voter's choice is easily automated (and thus the counting of all votes can also be done by machine). Such marksense systems have existed for decades. Within the domain of voting, the technology is often referred to as optical-scan becuase the marks are scanned using optical reading techniques.

[17] The machines are no longer used today in the USA, except in a very small number of constituencies where voters and election officials continue to trust their accuracy.

[18] A chad is the piece of paper that is left over after an attempt has been made to punch a hole in a ballot.

[19] The choice is often recorded by the voter filling in regular shapes, like rectangles, circles, or arrows.

As we have already noted for punch-card technology, humans are not machines and one should not rely on them to mark their ballot in a way which can always be scanned and correctly interpreted by a machine. The obvious solution to this problem is the one which has been adopted by many modern optical scan voting machines[KMR+07b] — use a computer interface to permit the voter to make their choices and use a machine to print out their marked ballot in a precise manner. However, this approach means that there are many more electronic components (hardware and software) in the voting machines. The use of such e-voting machines has led to many more interesting questions being asked about the trustworthiness of software and the quality of the software engineering processes being used in their development.

4.3.5 Direct-recording electronic (DRE) voting machines

Once we have a machine that can directly mark ballots on behalf of voters then there is a natural progression to removing the need to print the ballots — why not record the votes electronically and count these votes? It would appear — at first — that printing out the votes and then scanning them in to be counted introduces a redundant and unnecessary transformation of the vote data (from electronic screen, print to paper and scan back to electronic store). By removing the print and scan steps we arrive at direct-recording electronic (DRE) voting machines, which collect and tabulate votes in a single machine. Such machines are used by all voters in all elections in Brazil, and also on a large scale in India, the Netherlands, Venezuela, and the United States.

4.3.6 E-voting machines: paper trails

The current controversy surrounding DRE voting machines is that they are neither trustworthy nor trusted. Given a malfunctioning machine, can election officials fix the problem and correctly count the voters' intentions? Are the machines secure against attack — could the fairness of the election process be "compromised" in favour of a specific candidate? Returning to the initial requirement for secret ballots, how can a voter be sure that the machine does not provide any mechanism for a third party to find out how they have voted?

There has been a general call for the introduction of paper trails; but experts in the field of e-voting disagree on their practical implementation, particularly on the process of recounting the votes[YB08]. One of the most popular suggestions is to return to the optical-scan technology so that ballots can be counted by hand (if necessary). This return to paper ballots may appear to be ironic since the original voting machines were introduced in order to remove paper from the process. Another popular idea is to provide voters with receipts but this provides an added complication of meeting the requirement for a secret ballot. Typically, solutions to these apparently conflicting requirements involve complicated cryptographic protocols[Cha04].

We must ask why engineers, using mechanical technology over a century ago, were able to produce paper-less, trustworthy and trusted voting machines; but modern professional engineers, using electronic and software technology, are having difficulties in doing so. This thesis partially answers this question by analysing the important contribution of education in the engineering process.

4.3.7 Remote Electronic Voting — using a (public) network — a problematic future?

Remote electronic voting and postal voting share many properties in common, they both:

- enable voters to record their vote (remotely) without having to go to an official polling station,

- rely on a communication mechanism (which may be public) for transporting their remotely cast votes to a central location for counting, and
- depend on the communication mechanism to be trustworthy in order for their votes to be kept secret and unchanged.

There has been serious electoral fraud, throughout the world, due to postal voting. The variety of protocols that have been imagined for having a secure transport of postal votes is matched by the number of ingenious ways in which these protocols can be attacked. In truth, the postal voter is always required to exhibit some type of blind faith in the security of the underlying transport mechanism and communication protocols.

In general, the risk fo fraud due to postal voting must be weighed against the main potential benefits:

- for voters who are unable (or have great difficulty) to get to a voting station — perhaps due to some physical handicap — then offering a postal vote can be argued to give a fairer election process,
- the cost of requiring voters to attend a central polling station is considerable when compared with the cost of transporting their votes,
- the ease of voting by post — in comparison with having to go to a polling station — is likely to improve voter turnout,
- postal votes may help to prevent manipulation of elections through "get out the vote" attacks which involve transporting specific subsets of voters in a community to vote at a particular polling station, but deliberately not helping to transport other subsets.

Engineers of remote electronic voting systems have much to learn from analysis of postal voting.

Firstly, it is noteworthy that postal voting is not offered by all countries. In particular, when postal voting is not available it is often replaced by a mechanism called proxy voting - whereby a voter unable (or unwilling) to attend a polling station is able appoint someone as their proxy. This proxy is authorized by them to cast, or secure, their vote on their behalf. There are many different mechanisms for proxy voting, many of them differing on how they manage the case where a voter wishes to override their proxy. However, like remote voting, they require a trustworthy communication mechanism (in this case the proxy) to guarantee secrecy and accuracy. In some countries it is common for people to nominate an official of their chosen party as their proxy. In this way they can better trust that their vote is accurately communicated and recorded by their proxy. However, through such a choice there is a greater risk that they are unable to keep their vote secret. Consequently, such a mechanism is open to abuse and increases the risk of voter coercion. In other countries there is a limit (usually 1 or 2) to the number of proxy votes that can be authorized to an individual.

Secondly, a better understanding of remote voting requirements would be achieved by analysis of different postal voting mechanisms: are they different because they are meeting different requirements, or because they are implementing the same requirements in different ways? For example, in some states in America postal ballots can be delivered directly to an election official (or drop box) instead of the remote voter having to use (trust) the standard postal service. In some countries postal voting is on-demand (with no reason needed for authorising a postal vote), in other countries it is authorised only in restricted circumstances; and in other countries it is compulsory. Thus, very different approaches to postal voting have arisen even though in each instance there is general agreement that the election process must be both secret and accurate.

Finally, through analysis of problems with postal voting engineers would get a better understanding of the risks involved in remote electronic voting. In particular many problems occur in postal voting where the underlying

communication network is unreliable, for example: votes being lost in the public post. Perhaps this is an indicator of the biggest threat to using the public internet for transmission of ballots and votes — denial of service attacks[JRSW04a].

Despite major concerns with respect to security[Rub02], Internet voting systems have gained popularity around the world[KTV07]. Internet voting can facilitate voting from any internet capable computer, or can use traditional polling stations with voting booths consisting of internet connected computers. There are many disadvantages associated with the former[JRSW04a] and many advantages associated with the latter[SW08].

4.3.8 E-voting: success stories

Before we begin a review of the published scientific analysis of e-voting systems, it should be noted that e-voting has had public approval in a number of countries where there is a general perception of the successful adoption of modern voting technology. The move to e-voting can, in general, be justified by the following motives:

- Reduced costs
- Better and fairer accessibility
- Problems with previous technologies: trust and reliability
- Increasing Turnout
- Speed of count

There are clear examples of countries where the e-voting technology can be judged to be a success simply because of the specific underlying motives for it being adopted.

India

Indian E-voting machines were first manufactured in 1989-90 and were used on experimental basis in real elections in 1998. They were adopted across the whole population (greater than 500 million voters) in 2004. The motivation for adoption of e-voting technology in India were quite clear:

- Traditional voting was very expensive with respect to transport, storage, printing and operational costs. E-voting machines, although requiring considerable initial investment, have significantly reduced costs.
- Traditional voting systems required 30-40 hours to return the count. The vote-counting with e-voting machines is significantly faster, requiring 2-3 hours on average.
- Illiteracy in India is a significant problem and the e-voting machine interface simplified the process of selecting a candidate for illiterate voters.
- Vote stuffing was a problem with the traditional ballot boxes. The e-voting machines were programmed to record only five votes in a minute, which frustrated the bogus voters. (Although, even with this additional safety mechanism, there have been reports of electronic ballot box stuffing in India[Lau04].)
- When paper ballot systems were used in India, there were elections where the number of invalid votes was more than the winning margin between the candidates. Invalid votes can be avoided by use of e-voting machines.

Thus, e-voting machines in India can be judged a success based on the motivation for their adoption. There were Indian academics who raised concern about the trutsworthiness of the machines and whether they increased the risk of corrupt elections. However, there was a general public trust, and pride, in the systems and system developers. The main lessons to be learned from the largest democracy in the world are to keep the machine as simple as

possible, to own the system that is developed (through state ownership of the companies that manufacture the machines) and to take one's time in adopting the technology so that it can be well-tested before being applied across the whole country (involving half a million machines).

Estonia

The plan to introduce e-voting in Estonia was first announced by Parliament in 2001. Six years later, in 2007, Estonia became the first country to use internet voting in parliamentary elections, where approximately 3% of voters (30,000 in total) recorded their vote on-line. The system had already been tested nationwide in municipal voting in 2005, when 10,000 people cast their votes remotely.

Internet voting is available during an early voting period of a number of days before the election. Voters can change revote an unlimited number of times, with the final vote being counted. The internet vote is also overwritten by recording a vote at a polling station. The ability to revote temporarily caused some legal dispute over the principle of "one person, one vote" where problems arise if one inteprets the word vote to correspond to the ballot being cast by a voter.

The Estonian people appear happy with the system which uses already existing chip-and-pin identity cards in order for voters to be authorised to vote on-line.

The success of the Estonian experience is summarised in a report by Madise and Martins[MM06]:

> "Estonian e-voting experience seems to prove that it is possible to solve the legal as well technological obstacles. ...The system of e-voting has worked perfectly, all procedures have been legitimate and performed lawfully (respective confirmation of auditors is available).
>
> The attitude to the e-voting of the Estonian public was and is positive. There were no court cases and we do not have any information about purchase of e-votes (on the contrary to the votes on paper-ballot). Here we should underline again, that voting in privacy in the remote unsupervised Internet voting context is a right, not a duty."

The main lessons to be learned from the perceived success of remote electronic voting, by the voters in Estonia, is that familiarity with technology can increase trust. Estonia is acknowledged to be leading the way in e-governance and e-democracy, with wide use of government e-services by nearly half of households at home and and through public access. It is also the first (and currently, only) country in the world with compulsory pin-and chip ID cards with binding digital signatures. Remote e-voting is therefore seen as just another e-government service provided for the convenience of a mature "information society".

Switzerland

In Switzerland, the legal basis for the use of remote e-voting was established in 2002. In 2004 and 2005, five e-voting pilot trials were successfully executed. It should be noted that before these official trials, each of the three different electronic voting systems under consideration was subjected to rigorous evaluation by independent experts. Perhaps the most important lesson to learn is that the Swiss government had a very clear requirement that the remote e-voting systems should be at least as secure as voting by post[BB06], even though it was being offered as an alternative to postal voting rather than as a replacement. This baseline benchmark is much easier to pass than one which would require a system as secure as that offered by traditional polling booths. Thus, it should not be a surprise that the pilot trials led to the conclusion that remote e-voting was a success. Another aspect that should be taken into consideration is the motive behind these trials: about 10% of Swiss nationals live outside Switzerland, of which more than 120,000 have registered to vote. This means that remote voting can play an important role in deciding the results of Swiss elections.

4.4 Related Work

The related work which is most relevant to this project is presented in 4 sub-topics: E-voting, Software Product Lines, Event-B and Educational Research. There is — as would be hoped from the proposal objectives — interesting overlap between these topics.

4.4.1 E-voting

4.4.1.1 Analysis of systems: problems and risks

In 1990, in perhaps the first such article published in a mainstream computing journal, Peter Neumann analysed the *Risks in computerized elections*[Neu90]. In this article he acknowledges the important contributions of Ronnie Dugger[20] and Roy Saltman[21], who had already warned about risks in e-voting systems years before. Neumann's conclusions in 1990 are a good reflection of the current situation, almost 20 years later:

> "Providing sufficient assurances for computerized election integrity is a very difficult problem. Serious risks will always remain, and some elections will be compromised. ... we must question more forcefully whether computerized elections are really worth the risks, and if so, how to impose more meaningful constraints."

In 1992, Rebecca Mercuri, reviewed the risks involved in election systems and identified the need for rigorous verification if such systems are to be trusted[Mer92]:

> "It is incumbent on us to devise methodologies for designing verifiable systems that meet these stringent criteria, and to demand that they be implemented where necessary. "Trust us" should not be the bottom line for computer scientists."

A year later, she identified possible instances of *corrupt polling* in elections held in 1992[Mer93], and noted that:

> Technology alone does not eliminate the possibility of corruption and incompetence in elections; it merely changes the platform on which they may occur"

In 1998, Susan King Roth — who had previously examined ballot design for mechanical voting machines — warns about poorly designed e-voting systems[Rot98] which run the risk of disenfranchment:

> "Regardless, results indicate that greater attention to the usability and accuracy of voting systems during the development and evaluation stage would raise awareness and prevent "disenfranchisement by design"".

This article is also one of the first to mention evaluation criteria for comparing electronic and traditional voting systems.

In 1999 Larsen[Lar99] analysed two cases of electronic voting implementations which had major techical problems due to software errors. This is one of the earliest published articles in which strong evidence of "simple programming errors" in voting systems is reported.

[20] Dugger wrote a prescient article on the dangers of computerized vote-counting in The New Yorker of November 7, 1988.

[21] Whilst working for the National Institute of Standards and Technology in the USA, Saltman wrote two seminal reports on e-voting. The first, entitled *Effective Use of Computing Technology in Vote-Tallying*, was published in 1975 as **NBSIR 75-687** and re-published as **NBS SP 500-30** in 1978. The second report, entitled *Integrity, Accuracy and Security in Computerized Vote-Tallying*, was published in 1988 as **NBS SP 500-158**.

In 2001, Lorrie Faith Cranor wrote — in response to the problems in Florida during the 2000 Presidential elections — about the risks that arise when adopting new technology to replace existing voting systems[Cra01]. She highlights the importance of requirements, but her call for caution was clearly not heeded:

> "It is my hope that states will proceed cautiously in adopting new voting technologies, first establishing detailed requirements and certification criteria, and rigorously evaluating each candidate technology to see whether it meets the criteria. "

A year later Gritzalis[Gri02] — in *Principles and requirements for a secure e-voting system* — identifies the principle of coercion-free elections as being almost impossible to guarantee with internet voting:

> "It appears that certain requirements posed by legislation (e.g. uncoercibility) are really difficult, if at all possible, to be met with by the existing technology."

He concludes by stating that e-voting should be considered as complementary to traditional voting and that it should be introduced gradually.

Published at the same time, *Security Considerations for Remote Electronic Voting over the Internet* by Avi Rubin identifies risk of remote voting via the internet:

> "We conclude that at present, our infrastructure is inadequate for remote Internet voting."

In the same year, Mercuri writes about fundamental problems in e-voting. In *A Better Ballot Box?* she identifies the key issue of conflicting requirements between secrecy and auditability[Mer02b]:

> " As it turns out, many of the voting products currently for sale provide less accountability, poorer reliability, and greater opportunity for widespread fraud than those already in use. These problems result from an underlying fundamental conflict in the construction of electronic voting (e-voting) systems: the simultaneous need for privacy and auditability."

A month later, Mercuri reports that problems with the touchscreens of e-voting systems in Florida in September should have been avoided because similar problems had already occured in March[Mer02a]. She notes that there were many major issues with election systems development and testing procedures. In particular, she highlights the legal complications that can arise when one cannot trust election systems:

> "During court proceedings, it was revealed that Sequoia had sold the systems under trade-secret protection, making it a third-degree felony for Supervisor LePore if any details regarding the specification or internal functioning of the devices were revealed. Circuit Court Judge John Wessel granted Danciu a walk inspection of the voting equipment, where it was discovered that the pre-election testing circumvented the ballot face and the touchscreen was used only to cast one vote for each candidate listed first in every race. Because Danciu appeared third in his race, there is no test data that can reveal whether or not the machines would properly activate and record votes cast for him."

In 2003, McGaley and Gibson[MG03] analyse the introduction of e-voting in Ireland and recommend that e-voting should be considered as safety critical for the purposes of development, verification and maintenance. This view is supported in *Hack-a-Vote: Security Issues with Electronic Voting Systems*, where there is a reference to using formal methods techniques that are most commonly found in the development of critical systems[BPR+04]:

> "Auditing is not the only way to discover and patch security holes. Techniques such as proof-carrying code and system-specific static analysis can uncover specific vulnerabilities. A rigorous software engineering process also can help prevent the malicious introduction of security vulnerabilities."

This paper is also interesting as they experiment with deliberately hiding bugs in voting system software which prove very difficult to find using standard testing practices:

> "... it's easy to compromise a purely electronic voting system and difficult for auditors to identify and correct hacks that might otherwise completely compromise election results "

In 2004, McGaley and McCarthy identify a fundamental conflict between democratic and commercial interests in the development and adoption of e-voting technology[MM04]. They argue that transparency is a key factor in ensuring that commercial interests do not compromise the democratic process:

> "Transparency is an integral part of the security of voting systems. It is vital that technology is not allowed to erode that transparency. Not only must the technology itself implement measures to ensure that it is trustworthy ... but the system must be managed in a transparent, non-partisan way."

In *Code of Elections*[MC04] another fundamental conflict is identified:

> "The disparity between the code of election law and the code that comprises election equipment reflects inherent problems in the translation of social policies into computer procedures and overseeing processes. "

Whilst acknowledging that analysis of security from a technological viewpoint was critical, Xenakis and Macintosh argued that procedural issues were equally important for trustworthy e-voting[XM04b, XM04a]:

> "Our research has established the need for procedural security measures while at the same time demonstrated that the existing procedural safeguards are insufficient. Procedural security is directly applicable to procedures where the human factor is involved. However, since we are in a phase of re-defining the electoral procedures, we must primarily re-define the procedural responsibilities of the agents involved "

Despite previous published warnings about internet voting, the American government had plans to trial such a system — named SERVE (The Secure Electronic Registration and Voting Experiment) — for overseas military voters in the national elections of 2004. Consequently, a group of academics submitted a comprehensive report which led to the Pentagon cancelling the SERVE program. A summary of their report is published in [JRSW04b]. This report makes reference to previous results[KSRW04] where there is a comprehensive analysis of source code found in an e-voting machine that had a significant share of the market at the time. The main problems identified — " including unauthorized privilege escalation, incorrect use of cryptography, vulnerabilities to network threats, and poor software development processes " — lead to the authors supporting the call for a "voter-verifiable audit trail" in all e-voting systems.

In [KS04] Kocher and Schneier take an interesting view with respect to the cost and benefit analysis of compromising ("rigging") an e-election. Through simple analysis of the election system in America, they conclude that:

> "... voting systems must be designed to counter very well-funded and sophisticated opponents, including those with massive financial resources and the ability to join design teams, infiltrate manufacturing facilities, fabricate malicious integrated circuits, tamper with compilers, and mount a wide range of other attacks."

This conclusion is supported by Lauer[Lau04] where a generic threat model of e-voting procedures is proposed. However, their analysis was not consistent with the report — by Williams and King[WK04] — on the use of e-voting machines in Georgia (USA) in 2004. In this paper they conclude, controversially, that the introduction of e-voting was successful because it reduced the rate of undervoting from 4% to 1%.

In 2004, the Estonian government was in the process of adopting remote e-voting (REV). In contrast to America, the outlook for remote voting in Estonia was positive[Maa04]. Steps towards remote e-voting were also being taken in the United Kingdom. However, as reported by Storer and Duncan[SD04], there are limitations of cryptographic REV schemes such as that was being proposed for use in th UK at that time.

By 2005, academics from different disciplines were combining their expertise in order to analyse different aspects of specific voting machines. Initial reports focused on usability issues, mainly because such analysis was the easiest to carry out as it required no access to internal components of the systems. For example, in *Early Appraisals of Electronic Voting*[HBL$^+$05], the Diebold AccuVote-TS system is reviewed with respect to usability, and it is concluded that there are:

> "... reasons for optimism and some cause for concern".

The importance, and value, of such an article is not necessarily in the depth or rigour of their experimentation and analysis; it is in the objective scientific approach that is taken to their work. Storer and Duncan continued their work in analysing the problem of e-voting system adption in the United Kingdom[SD05], placing emphasis on security requirements and the need to improve current voting systems (at polling stations) as a means of increasing the public's trust before introducing remote voting. We note also their call for the application of formal modelling methods:

> "Whilst a plethora of requirements documents continue to be produced for electronic voting technologies, a formal basis for their development would be more satisfactory. "

Xenakis and Macintosh add to the analysis of e-voting in the UK by examining the additional problem of administrating elections using e-voting technology[XM05a]. They conclude, unsurprisingly, that:

> "Systematic staff training in the administration of the new methods of voting is also required to support the operation of the e-electoral process"

What is surprising is that in elections following their analysis there are many reports of problems occuring because of poorly prepared election staff. In a complementary paper published in the same year, Xenakis and Macintosh examine the need for social acceptance for the new e-electoral practices, and emphasise the role of security procedures in achieving acceptance[XM05b]:

> "A voter's perception of security of the electoral process is equally important to the actual security itself. Since procedural security is evident and understandable to voters, it has a comparative advantage when it comes to developing and supporting the social acceptance for the new e-processes."

In 2005, the adoption of e-voting in Ireland was seen as a government failure. Questions arose as to whether such failures were common to information systems; and, if so, could a reasonable explanation be found. Zelic and Stahl examine the problem from the point of view of ontologies[ZS05]:

> "...the ontological view of an information system is the root cause for many of the problems it encounters and that the Irish evoting experience is a good example of this."

In the following year, Braun and Brandli report on the success of e-voting trials just finished in Switzerland[BB06], whilst highlighting the need for future vigilance.

> " Secure e-voting is feasible: the pilot trials have demonstrated this. But ongoing security depends on being able to maintain control of continually changing threats and risks. The necessary security measures cannot be developed and put in place once and for all."

They also identify the engineering compromises that must be taken with regard to degrees of security in e-voting:

> "The challenge therefore lies in providing the greatest possible degree of security at an affordable price. At the same time, user-friendliness must not be excessively restricted. "

Also in 2006, a year after Estonia was the first country to implement internet voting in a national election, there is general public support for the system[MM06]. However, we note that: "Approximately 2% of actual voters made use of this possibility."

By 2006, many different protocols and schemes for secure and verifiable e-voting had been proposed. In *Kleptographic Attacks on E-Voting Schemes*[GKK+06] it is argued that a wide-range of such schemes are amenable to a specific sort of attack, which had been overlooked in the original designs. This paper is a good example of the problems in adopting new technology to meet voting requirements: when can we be sure that we haven't overlooked some weakness that could appear after the machines have already been procured, developed or deployed? Furthermore, such results — although valuable to researchers and developers in the domain of e-voting — will inevitably undermine the public's trust in the adoption of e-voting technology.

Another problem was becoming more evident in the media's attention on e-voting: there was widespread disbelief at the lack of professionalism in the engineering of the e-voting systems that had been analysed. Don Gotterbarn summarises the problem in [Got06]:

> "...the development of the systems by e-voting companies seems unguided by any application of IT professionalism. The professional develops systems which take into account the interests of all of the stakeholders, understand the social, political, and ethical issues. The absence of these considerations leaves us with failed systems. This kind of failure not only harms the voting process but it harms the practice of IT if we are not vocal about how these systems are professional failures. "

In particular, this issue has arisen in all independent analyses of e-voting system software.

Researchers continued to analyse risks in e-voting technology and identified a major issue regarding the verification and certification of e-voting systems that incorporate commercial-off-the-shelf (COTS) components[MLF06]:

> "...all versions of the federal voting system guidelines exempt COTS hardware and software from inspection ...This loophole is anathema to security and integrity"

In effect, the exception goes against good engineering practice and even though the academic community had tried to close this loophole there has been no move towards a chnage in legislation. A second problem arises when manufacturers try to exploit this loop hole:

> "...Diebold Election Systems had erroneously reported to a testing authority (CIBER) that certain Windows CE operating system files were commercial-off-the-shelf (COTS) but in fact also contained customized code."

In *E-Voting in Brazil — The Risks to Democracy*[RFAB06] we see evidence of mistrust in voting technologies moving across national boundaries:

> "...Literature has shown that countries with strong democratic traditions, such as the United States and Canada, are not yet using electronic voting systems intensively, due to the concern for and emphasis on security ...the introduction of e-voting in Brazil is highly risky to democracy due to the lack of emphasis on security and the lack of a socially informed and socially driven approach to technological innovation"

In 2007 there was a large growth in the number of academic (research) papers published concerned with analysis of e-voting machine risks and comparisons with traditional voting systems. It became clear that in order to better understand the issues surrounding e-voting it would be necessary to get a better understanding of voting in general. In [BGE07] usability issues for e-voting are addressed by first comparing the usability of different traditional voting systems. Thus type of analysis is fundamental in building a domain model and for informing the specification of e-voting requirements. One insight from their research is that usability is not just an issue for voters it is also important for the other users of the system, namely the election officials:

> "...This raises the more general issue of usability of procedures and technologies not just for voters, but for poll workers and other election administrators as well. Poor usability may influence the accuracy and security of post election activities such as tabulating and canvassing. This is a largely unexplored problem."

A thesis by Everett[Eve07] analyses *The Usability of Electronic Voting Machines and How Votes Can Be Changed Without Detection*. This work concludes that "...there are not differences between DREs and older methods in efficiency or effectiveness. However, in terms of user satisfaction, the DREs are significantly better than the older methods". However, the thesis demonstrates that "...over 60% of voters do not notice if their votes as shown on the review screen are different than how they were selected" and thus "...malicious software installed on a DRE could steal votes right in front of voters with a low probability of being detected". Jefferson [Jef07] argues — through analysis of undervoting in Florida 2006 — that poorly designed ballots (as represented at an electronic interface) probably resulted in the wrong candidate from being elected. In this case there is no suggestion that the poor interface resulted from malicious code or by malicious design; however it reinforces the ease with which one could manipulate an election if one controlled the electronic interface (hardware and/or software).

One of the most influential papers on e-voting machines reports on vulnerabilities of the Diebold AccuVote-TS Voting Machine which are in widespread use across America[FHF07]. They conclude, based on their analysis, that:

> " DREs may resist small-scale fraud as well as, or better than, older voting technologies; but DREs are much more vulnerable to large-scale fraud."

In another paper, further analysis of the Diebold Voting Machine examines whether it is possible to have a mechanism by which a poll worker, on election day, could validate that the software in the voting machine is the software that was produced by the vendor, without modification. They go on to:

> "...demonstrate that the current state of the art in software-based attestation is not sufficiently robust to provide humanly verifiable voting machine integrity in practice."

Given the difficulty in validating that software on voting machines, another approach is to ensure that the machines are secure and that access control would mitigate an attack involving installation of new software. However, a study by European researchers examined the Nedap/Groenendaal ES3B voting computer[GH07], and it shows that the installation of new software in Nedap ES3B voting computers is straightforward:

> "...anyone, when given brief access to the devices at any time before the election, can gain complete and virtually undetectable control over the election results".

A similar study showed major vulnerabilities in machines based on optical scan technology[KMR+07b] and outlined how the correct functioning of the AccuVote Optical Scan voting terminal (AV-OS) manufactured by Diebold could be compromised[KMR+07a].

It was now becoming clear that all e-voting systems had vulnerabilities that made them open to attack. An interesting question is how, after identifying potential attacks, one can introduce procedures for election officials in order to block or mitigate them. Analysis of the Hart Intercivic DAU eSlate (an e-voting system enabled for disabled users)[PRH+07] demonstrates that the human element is critical:

> "...safety depends on informed procedures being devised and followed. .... People, even election officials, make mistakes. People can be compromised, often unknowingly. Procedures — for elections involving electronic voting systems, and not involving electronic voting systems — must take human imperfections into account."

By 2007, e-voting machines, in America, should have been accredited against legal standards. However, one must question whether standardisation procedures are working. With this in mind, Ryan and Hoke analyse the Diebold Election Systems, Inc. election management software (GEMS)[RH07] and they found that "the GEMS architecture fails to conform to fundamental database design principles and software industry standards". This not only reflects badly on the manufacturers but also on the quality of the standards:

> "Despite these technical and systemic deficiencies, GEMS received approval as complying with Federal Voting System 2002 standards. Questions then arise concerning the adequacy of the 2002 and 2005 regulatory standards ... the standards structurally encourage and reward election system vendors for using less exacting database design standards."

Analysis of e-voting in Ireland — where the vote counting (tabulation) algorithm is very complex — identified the potential for errors in the vote counting process. Kiniry proposes that the counting process be developed formally and demonstrates the feasibility of such an approach[Kin07].

Voter verified audit trails — involving paper records — are, in 2008, one of the most popular solutions to the problem of trusting an e-voting system to correctly count votes. However, as reported in *Evaluating Electronic Voting Systems Equipped with Voter-Verified Paper Records*[ASH+08], it is important to "evaluate a printer's performance and its integration with the overall voting system". The introduction of a print functionality is not

straigtforward and involves additional risks that need to be managed. We note that this work arose out of an official request by the Attorney General's Office of New Jersey, who issued criteria for e-voting machines equipped with printers and procured testing of various systems against these criteria.

The paper *Security Evaluation of ES&S Voting Machines and Election Management System*[ACC+08] summarizes a security analysis of the DRE and optical scan voting systems manufactured by Election Systems and Software (ES&S), as used in Ohio (and many other jurisdictions inside and outside the US). Their analysis identifies:

> " ...numerous exploitable vulnerabilities in nearly every component of the ES&S system. These vulnerabilities enable attacks that could alter or forge precinct results, install corrupt firmware, and erase audit records."

We note that this work arose out of participation in official reviews of e-voting systems in Ohio[22].

In *Are Your Votes Really Counted? Testing the Security of Real-world Electronic Voting Systems*[BBC+08] a review of security testing for e-voting systems evaluates not only the machines but the way in which the machines are tested:

> " Our experience suggests that there is a need for a drastic change in the way in which electronic systems are designed, developed, and tested. Researchers, practitioners, and policy makers need to define novel testing approaches that take into account the peculiar information flow of these systems, as well as the combination of computer security mechanisms and physical procedures necessary to provide a high level of assurance."

We note that this work arose out of participation in official reviews of e-voting systems in California[23] and in the EVEREST project in Ohio.

A review of the EVEREST project is found in *Systemic Issues in the Hart InterCivic and Premier Voting Systems: Reflections on Project EVEREST*[BEH+08]. This paper proposes that future analysis of e-voting systems should start with analysis of known vulnerabilities, as this will naturally lead to the discovery of previously inknown weaknesses:

> " ...by forcing ourselves to begin with the confirmation of known vulnerabilities, we were able to quickly learn about the innerworkings of the Hart and Premier systems. This process not only added value to the community by providing independent validation of previously known problems, but also served to help us quickly identify new vulnerabilities in both previously evaluated and new components of the system."

In *Pre-Election Testing and Post-Election Audit of Optical Scan Voting Terminal Memory Cards*[DKK+08], the issue of risks arising out of customizable components, in optical scan machines, is raised:

> "Optical scan electronic voting machines employ software components that are customized for each specific election. Such software components are critical from a security and integrity point of view, as they define ballot layout and outcome reporting facilities. The possibility of these components to be tampered with presents a major concern as incorrect election results may be produced due to either malicious interference or accidental corruption."

[22] Ohio's Evaluation & Validation of Election-Related Equipment, Standards & Testing (EVEREST) in December 2007
[23] California's Top-To-Bottom Review (TTBR) in July 2007

We note that this work arose out of an official request from the Office of the Secretary of the State of Connecticut.

In 2008 there was continuing analysis of e-voting in a European context. For example, *Modeling and Analysis of Procedural Security in (e)Voting: the Trentino's Approach and Experiences*[WV08] reports on research carried out as part of the ProVotE project[24]. An interesting aspect to their work is that the approach they take is "based on providing formal specifications of the procedures and using model checkers to help us analyze the effects of attacks"

At the same time as e-voting systems were being analysed, and weaknesses being identified, there were increasing numbers of calls for paper trails and human auditing. It became clear — as reported in *Improving the Security, Transparency and Efficiency of California's 1% Manual Tally Procedures*[Hal08] — that analysis of manual counting procedures would give rise to a better understanding of "how voting systems could better support manual tally audits". Thus, the problem of voting systems seems, in 2008, to have turned full circle. However, closer inspection of research in specific subdomains of e-voting (as reviewed in following sections) shows that real progress has been made.

4.4.1.2 Architecture: design and implementation

In 1998, Susan King Roth identified voter disenfranchisement as a main risk of poorly designed e-voting systems[Rot98]. Although her analysis focussed on design issues related to man-machine intrefaces, her worked raised three interesting questions with respect to poorly designed machines, in general:

> "...if systems are found to cause an unacceptable error rate resulting in rejected ballots, as evidenced by problems with punch cards in some districts in the U.S., does this imply that previous election results can be challenged by losing parties? Should there be national supervision and certification of voting systems rather than the local supervisory structures now in place, which means information may not be shared across state lines? Can public confidence in the voting process be maintained if problems with voting systems are published in the media?"

In 2002, Mercuri analysed the problem of designing a *A better ballot box*[Mer02b]:

> " Despite manufacturers' statements to the contrary, it is beyond the scope of present computer science and engineering principles to design a fully electronic, self-auditing voting system that sufficiently guarantees that all ballots are recorded and tallied in accordance with the voters' intentions. "

In 2003 the design of an internet voting system is proposed in *REVS Ű A Robust Electronic Voting System*[JZF03]. They write that they have designed:

> "a robust electronic voting system ... that tolerates failures in communications and servers while maintaining all desired properties of a voting system. Another important characteristic of REVS is the ballot independency; which facilitates its use in any kind of elections or surveys. The implementation of REVS was carefully designed for assuring scalability and availability in large-scale elections."

Thus, it would seem that internet voting was not so problematic after all. However, the key issue of anonymity is mentioned only briefly in the conclusions, where the authors state that "REVS can beneficiate from a more sophisticated anonymity mechanism".

Similarly, in 2004, Selker and Goler report on *The SAVE system — secure architecture for voting electronically*[SG04]:

[24] A four years project sponsored by the Autonomous Province of Trento that has the goal of switching to e-voting for local elections.

> " This voting architecture provides a means to vote over open networks in a way that is reliable, secure, and private. Due to its modularity and common specifications, it is easy to implement, improve and it is inexpensive. The system uses COTS equipment for the all of the back-end systems, reducing the likelihood of fraud with the system components as well as keeping the cost down."

Their proposal is based on demonstrating that — through n-version redundancy techniques — there is no single point of failure in their system. It is possible that such an approach will lead to a more trustworthy system but it is not clear that the additional complexity will make it easier for voters (and other users) to trust. Their architecture complicates the issue of voter verifiability; but they do state that they "are also examining ways of providing verifiable feedback to users, but in a way that does not compromise the confidentiality and receiptfreeness requirements of voting".

In 2004, further analysis of the REVS architecture identified weaknesses inherent in the design due to voter information being centralised[SD04]. By distributing voter information across different election authority domains, then no single authority (or person) can use the information it collects in order to violate key requirements such as secrecy and accuracy.

The notion of "Design for dependability" reappears in an article by Bryans et al. in 2006[BLRS06], where they consider the importance of robustness and fault-tolerance. They argue that a good design must ensure that:

> "... accidental or malicious corruption of votes will be detected. But these error detection mechanisms by themselves do not reduce the probability of failures. Detected errors must also be dealt with properly; aborted elections are still failures. "

In that spirit, the paper *Tamper-Evident, History-Independent, Subliminal-Free Data Structures on PROM Storage — or — How to Store Ballots on a Voting Machine*[MKSW06] proposes a design for an e-voting system component responsible for vote storage.

The notion of "Design for verification" reappears in an article by Sastry, Kohno and Wagner in 2006[SKW06]. Their approach uses hardware to isolate components in DRE machines, improving their security — specified as key properties — "without modifying the existing voter experience or burdening the voter with additional checks or procedures". An approach to designing the voter interface of a DRE machine, where simplicity is proposed as a means of assisting verification and testing, is outlined in *Prerendered User Interfaces for Higher-Assurance Electronic Voting*[YWHB06]. The paper presents "a specific design for a touchscreen voting machine" and demonstrates that "it can be implemented in a small fraction of the amount of code in current voting machines". We note more details of the research into Prerendered User Interfaces (PRUI) for e-voting is found in Yee's thesis published in 2007[Yee07a]; and that Yee extends this work to support other interface features such as accessibility[Yee07b].

The design of remote electonic voting machines — requiring a network for communication between machines — is a much more complex problem than the design of standalone machines. In *e-Voting Requirements and Implementation*[AFT07], they highlight "the complexity of the deployment of e-voting systems and the inherent security issues that arise from the underlying distributed system". They propose an architecture design that focusses on "the security of the election servers and the channels between client machines and the servers". Unfortunately, they identify a major weakeness in their architecture (and with remote voting, in general) — they cannot guarantee the security of the client machine from which a vote is cast. The design of a secure remote evoting system is also proposed in *Civitas: A Secure Remote Voting System*[CCM07]. The authors acknowledge that the problem is difficult:

> " Yet as hard as secure supervised voting may be, secure remote voting is even harder."

but go on to argue that "it is possible today to build a secure, practical, remote voting system". The use of the word "practical" is quite deliberate — the authors argue that prior work illustrates a conflict between security and practicality, but that their prototype "resolves this conflict by demonstrating for the first time that strong security properties can be offered by a practical remote voting system." The paper addresses one of the major problems with remote voting: how can one ensure that voters cannot be coerced when the voting location is unsupervised? In particular they use the requirement that "voters cannot prove whether or how they voted, even if they can interact with the adversary while voting." Their design — based on the formal model first proposed in [JCJ05] assigns a "private credential" to each voter in order for them to be able to vote. This permits each voter to produce a fake private credential which, to an adversary, is indistinguishable from a valid credential Thus, under coercion from an adversary, a voter can substitute a fake private credential for their real private credential. It should be noted that the architecture may be susceptible to denial-of-service attacks:

> "Civitas does not guarantee availability of either election authorities or the results of an election. However, the design of Civitas accommodates complementary techniques for achieving high availability."

Furthermore, responsibility now falls on voters to use a trustworthy client machine when recording their vote. It remains to be seen if such an approach is as practical as claimed: it certainly complicates the registration and voting process, and it is not clear whether the system for allocating credentials (real or fake) would be trusted by voters.

Qadah and Taha propose an alternative remote e-voting architecture and illustrate how mobile devices can be used as voting client machines[QT07]. However, they do note that their implementation — using public wireless networks — is not suitable for secure elections:

> "...for highly secure elections, such as political ones, voters need to access the e-voting system through secure channels including the use of secure client devices located at secure polling locations and connected to the e-voting system through secure Intranets/private networks"

The problem of voting machine integrity could be addressed by explicitly designing the system to be self-attesting. In *On the Difficulty of Validating Voting Machine Software with Software*[GGR07], the authors argue that "the current state of the art in software-based attestation is not sufficiently robust to provide humanly verifiable voting machine integrity in practice". They go on to design a self-attesting vote system based on current technology and best practice. Through analysis of risks and threats they then implement an attack on the system that indicates that self-attesting evoting systems (based on their architectural approach) are "currently impractical for use and ...as technology advances, the attack will likely become more effective".

Lundin contributes to the research on e-voting system architectures by proposing a component-based development and analysis framework constructed on a layered architecture[Lun07]:

> "...in order to build an e-voting system we simply add certain distinct pieces together - and in order to improve on a particular system we swap one distinct piece for another that fits into the same slot"

Sandler and Wallach take an innovative approach to developing e-voting systems, where there is a requirement for auditing the procedures followed and events encountered during system execution: they first express a generic set of auditing requirements and construct a generic infrastructure — called *Auditorium* — to meet these

requirements[SW07]. They then demonstrate how the Auditorium can be instantiated in order to meet specific e-voting auditing requirements. They acknowledge that their infrastructure does not — and was not designed to — address key aspects of e-voting such as trustworthy software and voter verificability; but they conclude that their "design can be added to electronic voting systems in use today, including those used in the election whose auditing anomalies inspired our work." This approach is complementary to component-based development and together they provide strong motivation for research into e-voting software product lines.

The paper *Security Evaluation of ES&S Voting Machines and Election Management System*[ACC+08] carries out a general analysis of security. An interesting aspect to their analysis was that by focusing on the architectural issues they identified the problem of unwanted interactions between components:

> "interactions between various software and hardware modules leads to systemic vulnerabilities that do not appear to be easily countered with election procedures or software updates."

This is noteworthy as interactions are a major problem in component-based software development and product line approaches.

In *Analysis of a distributed e-voting system architecture against quality of service requirements*[GLR08a], the issue of the quality of service provided by a distributed e-voting system is considered. The paper models different communication architectures and — through simulation — demonstrates that certain architectures are unable to provide acceptable quality of service (formulated as the time required to vote), given the requirement that voters are permitted to vote from any authorised voting station. We note that such a requirement is promoted, independently, by *The case for networked remote voting precincts*[SW08].

In *Verifiable Anonymous Vote Submission*[ZA08] adapt the *REVS* architecture[JZF03] to deal with anonymity and verifiability. This work is based on two previous anonymization architectures — Mix Nets and Mix Rings — which were not originally intended for e-voting systems but which now form the central design feature of many proposals for remote electronic voting.

4.4.1.3 Counting and Tabulation

The first published work specifically on using a computer to count (tabulate) votes was the paper *Algorithm 123 — Single Transferable Vote by Meek's Method*[HWW87]. The authors introduce a better algorithm for distributing surplus votes that is easily executed on a computer but which would not be suitable for human tabulation. After specifying the new algorithm — in natural language — they then go on to implement it in Pascal. Finally, they demonstrate — through manual proof — that the equations that need to be solved at each stage of Meek's method have a unique solution.

Mukherjee and Wichmann follow up this work by formally specifying the algorithm in VDM[MW93, MW95]. Two years later, Poppleton — in *The Single Transferable Voting System: Functional Decomposition in Formal Specifications*[Pop97] — uses the formal specification language Z to model the counting rules for a simplified form of STV. The paper demonstrates the utility of a formal specification for validation.

In 2000, the paper *Better voting methods through technology: The refinement-manageability trade-off in the single transferable vote* by Tideman and Richardson[TR00] argue that

> "...every refinement [to the STV algorithm] comes at a cost of increased difficulty of understanding the vote-counting algorithm and increased cost of undertaking the count."

They demonstrate that implementing different variations of count algorithms, in software, facilitates experimentation with refinements that would not be feasible with manual counting.

In *Preliminary Study to Empirically Investigate the Comprehensibility of Requirements Specifications*, Carew et al.[CEB+05] compare the validation of the legal definition of STV counting rules expressed in natural language and expressed in a programming language. They stress the importance of structuring the code so its structure matches the structure of the legal documents.

In 2007, the paper *Verification-Centric Realization of Electronic Vote Counting*[KCT07] reprots on the implementation of a vote count algorithm using the Java Modelling Language (JML) and demonstrates techniques for verification of safety properties expressed as preconditions, postconditions and invariants.

4.4.1.4 Formal Methods

The first reported uses of formal methods in the domain of e-voting were concerned with specifying the counting algorithms for alternative vote systems[HWW87, Pop97]. Both these early papers demonstrated the advantages of using formal methods to analyse complex vote tabulation procedures.

In 2003, McGaley and Gibson — in *E-Voting: A Safety Critical System* — proposed the use of formal methods for the synthesis and analysis of all critical components[MG03]. This work was in response to the Irish government's to introduce an e-voting system which had not been built following best practices for critical system development.

In 2005, Juels, Catalano and Jakobsson report on *Coercion-resistant electronic elections*[JCJ05]. They state that their major contribution is to: "describe and characterize a new and strengthened adversary for coercion in elections", and claim that they "additionally present what we believe to be the first formal security definitions for electronic elections of any type." They formulate the related properties of correctness — as when a computation of tally always yields a valid tabulation of ballots — and verifiability — as the ability for any player to check whether the tally has been correctly computed.

In the same year, Kremer and Ryan report on an *Analysis of an electronic voting protocol in the applied pi-calculus*[KR05]. The write "Recently highlighted inadequacies of implemented systems have demonstrated the importance of formally verifying the underlying voting protocols." They then demonstrate how the applied pi calculus could model a known protocol for elections; and "formalise three of its expected properties, namely fairness, eligibility, and privacy." It should be noted that their verfication was not fully automated, and required manual proof of some key properties. In a follow-on paper — *Coercion-Resistance and Receipt-Freeness in Electronic Voting* — a year later, Delaune, Kremer and Ryan studied two particular anonymity properties of election protocols: receipt-freeness and coercion-resistance. They showed that receipt-freeness can be expressed using observational equivalence from the applied pi calculus; but they needed to introduce a new relation to capture coercion-resistance.

In 2006, in *Formal techniques in a remote voting system*[KMC+06, Kin07, KCT07], the authors describe the formal techniques incorporated during the development of components of the Kiezen op Afstand1 (KOA) open source e-voting system for remote elections. In particular, they report on the use of the JML and the ESC/Java2 tool for verification of the count implementation in Java.

A year later, in *Evaluating Procedural Alternatives in an e-Voting Domain: Lessons Learned*[BFMV07], we see one of the first modelling approaches to mix formalisms in the specification of e-voting system behaviour. The authors present a modelling approach based on the integration of UML and Tropos, exploiting complementary features of the two modelling approaches and allows them to "maintain both an operational view of the voting procedures and a visual approach to evaluate choices in designing the electronic processes". The modelling is not formal enough to facilitate automated verification, but their combination of formalisms is noteworthy.

The same year, Cansell Gibson and Mery published complementary papers that illustrated how the B method could be used to verify voting system properties. Firstly, they considered the *Formal verification of tamper-evident storage for e-voting*[CGM07a]. Secondly, the reported on *Refinement: A Constructive Approach to Formal Software Design for a Secure e-voting Interface*[CGM07b]. General interest in *Verification and Validation Issues in Electronic Voting*[CC07] (and the need for rigour[Gib07] in order to achieve reliability[Yee07a]) was growing; and many examples of systems being evaluated using formal methods were being published. A good example is *Simulation-based analysis of E2E voting systems*[dMPQ07], where the authors use simulation to analyse properties of security protocols in different systems. The main motivation given is that:

> " End-to-end auditable voting systems are expected to guar- antee very interesting, and often sophisticated security properties, in- cluding correctness, privacy, fairness, receipt-freeness, . . . However, for many well-known protocols, these properties have never been analyzed in a systematic way. In this paper, we investigate the use of techniques from the simulation-based security tradition for the analysis of these protocols."

The general question of using formal methods in electronic governance (including e-voting) was raised in *Technological foundations of electronic governance*[DJOS07], where the authors discuss the "relevance and opportunities for the application of mature Formal Techniques — techniques based on mathematical theories and supported by industry-ready tools and methods — to build technical solutions for Electronic Governance".

In *An Information-Theoretic Model of Voting Systems[HV07], Hosp and Vora motivate an information-theoretic approach to rating voting systems for integrity, privacy and verifiability. They develop a mathematical framework and show that tradeoffs exist between integrity and privacy and between verifiability and privacy. This paper is interesting because it shows how mathematical modelling can be used in formalising and making design decisions.*

In 2008, the formal description technique Estelle is used in the Analysis of a distributed e-voting system architecture against quality of service requirements[GLR08a]. At the same time, the property elicitation tool PROPEL, toegther with the finite-state verifier FLAVERS, were used to formally verify that an election process model adhered to key properties[SMCO08]. Focusing on procedural security — the paper *Modeling and Analysis of Procedural Security in (e)Voting: The Trentino's Approach and Experiences*[WV08] — combined UML modelling with reasoning in temporal logic in order to understand the effect and impact of faults on safety requirements.

In 2009, the *First International Workshop on Requirements Engineering for e-Voting Systems[GJ09] included a number of papers on the use of formal methods.*

Recently, the paper Engineering a distributed e-voting system architecture: meeting critical requirements*[GLR10] illustrates the need for different modelling languages in the specification of e-voting architectural requirements:*

> *"Given the different roles played by the requirements and design models, we believe that there is a need for a number of different modelling languages when verifying designs against different types of requirements. Of course, this poses the problem of how to ensure that the different models are consistent and how to integrate them into a coherent whole"*

In Formal Specification and Analysis of an E-voting System[WKV10] the authors "use the ASTRAL language to specify the voting process of ES&S machines and the critical security requirements for the system. Proof obligations that verify that the specified system meets the critical requirements were automatically generated by the ASTRAL Software Development Environment (SDE). The PVS interactive theorem prover was then used to

apply the appropriate proof strategies and discharge the proof obligations."

4.4.1.5 Domain Modelling

By 2005, it had become clear that poorly engineered requirements was a major problem common to many e-voting systems. The main cause of this problem was that e-voting was a poorly understood domain and would benefit from a more rigorous modelling of the domain.

In *Does Ontology Influence Technological Projects? The Case of Irish Electronic Voting*[ZS05] Zelic and Stahl examine different types of ontological reasoning within the context of the e-voting problems that were reported in Ireland. As ontologies (and their modelling) are an important aspect of domain modelling, this paper — through higlighting the fact that different ontological viewpoints can give rise to different conceptions of technoogy — provides a warning that it is not sufficient to build an ontology in order to better understand the e-voting problem domain, one must also chose the right sort of ontology.

Another approach to modelling a problem domain, such as (e-)voting, is to propose a framework upon which any system that works in the domain can be evaluated. As such a framework evolves and is refined, it will implicitly contain domain knowledge. Transforming such an evaluation framework (such as that proposed by Sampigethaya and Poovendran[SP06]) is a good initial step in the construction of a domain model.

In *Evaluating Procedural Alternatives in an e-Voting Domain: Lesson Learned*[BFMV07], the authors propose modelling the procedural aspects of (e-)voting using a combination of modelling languages. They note that the domain is very complex and that future work is necessary in order to widen the domain (model) to incorporate aspects such a security. Furthermore, they note that the expressiveness of their underlying modelling languages and the limits of the tool support, for automated reasoning, significantly weaken their approach to domain modelling.

4.4.1.6 Engineering: general views on software quality

As early as 1990, Neumann commented on the risks of poorly engineered software in e-voting machines[Neu90]:

> "Vendors can hide behind a mask of secrecy with regard to their proprietary programs and practice, especially in the absence of controls. Poor software engineering is thus easy to hide. Local election officials are typically not sufficiently computer- literate to fully understand the risks."

Two years later, Mercuri comments on the fact that there may be a need for restrictions on the individuals who can engineer the software in e-voting systems[Mer92]:

> "... no laws ... presently preclude convicted felons or foreign nationals from manufacturing, engineering, programming or servicing voting machines. ... "Trust us" should not be the bottom line for computer scientists.

We should ask whether only professionaly qualified software engineers should be permitted to develop the software in e-voting machines. However, this question raises even more issues as no appropriate qualification currently exists.

A year later, Mercuri writes that software bugs had been reported for some (apparently) simple programming tasks such as merging vote counts[Mer93]:

> "Difficulties with the central software for merging the electronic and mechanical tallies created further delays in reporting results."

This early warning, concerning the poor quality of code that cannot guarantee correct functionality for the simplest of tasks, was largely ignored in later years.

In *Principles and requirements for a secure e-voting system*[Gri02], Gritzalis puts forward the proposal that the quality of the software in an e-voting system can be guaranteed by the process used in its development. Thus, a system that is developed using standard, well-accepted, engineering methods should be more turstworthy than a system that is not. (One could argue that professional engineers are very likely to use such methods in the development of e-voting systems, and thus we need only trust that the individual developers are properly qualified.) This theme of trust and secure engineering is addressed by many of the articles in the book *Secure Electronic Voting*(edited by Gritzalis) in 2003[Gri03].

McGaley and Gibson propose that formal methods may be the best approach to engineering the critical software that counts the votes[MG03]:

> "...it is perfectly realistic to assume that some mistakes may have been made in the development of the count software, especially since the system was not developed formally."

In Grove's review of the *ACM Statement on Voting Machines*[Gro04b] the issue of poorly engineered software is repeated several times, for example:

> "...many electronic voting systems have been evaluated by independent, generally recognized experts and have been found to be poorly designed; developed using inferior software engineering processes; designed without (or with very limited) external audit capabilities; intended for operation without obvious protective measures; and deployed without rigorous, scientifically designed testing.
>
> ...such systems must embody careful engineering ...
>
> "

Ten years later, the ACM communications published a special issue — *The problems and potentials of voting systems* — on e-voting[Neu04]. It is worthy of note that the need for properly engineered software was a recurring theme in the majority of the articles contained within.

McGaley and McCarthy return to the issue of professional engineering of software in e-voting systems in *Transparency and e-Voting: Democratic vs. commercial interests*[MM04]

> "No bridge would be built in the developed world without the involvement of an engineer, and yet computer systems are commonly installed by people with minimal knowledge and training."

Despite poorly developed software being a major problem in e-voting systems, many of the proposed security mechanisms depend on additional, complex, software functioning correctly. For example, this is a problem with using digital signatures[KSW05]:

> "...However, since voters cannot verify signatures on their own, this approach requires another set of hardware devices and software that voters must trust."

One must question whether an engineering approach that requires more software (as a critical system component) is appropriate in the current context of the way in which software is constructed by e-voting machine manufacturers.

Neumann writes about the *Responsibilities of Technologists*[Neu05] and notes the need for professional experts to be involved in the evaluation of e-voting systems. In particular, there is a pressing need for software expertise as there are an increasing number of reports of trivial voting functionality being correctly implemented (in software):

"Some machines lost votes because of programming problems, or recorded more votes than voters."

4.4.1.7 Cryptography: Protocols and Schemes for Anonymity and Verifiability

In 1981, perhaps the earliest reference to the use of cryptographic protocols in electronic voting appears in *Untraceable Electronic Mail, Return addresses, and Digital Pseudonyms*[Cha81] by Chaum. The paper presents a technique, based on mixing messages to ensure anonymous channels that separate votes from voters, that

> "...allows an electronic mail system to hide who a participant communicates with as well as the content of the communication — in spite of an unsecured underlying telecommunication system. The technique does not require a universally trusted authority."

The application of the technique to voter-verifiable elections is suggested:

> "Elections in which any interested party can verify that the ballots have been properly counted are possible if anonymously mailed ballots are signed with pseudonyms from a roster of registered voters."

In 1982, DeMillo, Lynch and Merritt analyse the security of *Cryptographic Protocols*[DLM82]. They introduce a general model which they specialise to different application areas, including "secret ballot elections". In the same year, Yao published *Protocols for Secure Computations*[Yao82] in which the use of one way functions is shown to be appropriate for implementing secret voting systems.

In 1985, Cohn and Fischer published a paper — *A Robust and Verifiable Cryptographically Secure Election Scheme* [CF85] where the notion of an e-voting system being robust against attacks is introduced:

> " ...it is robust in the sense that no conspiracy of dishonest voters can prevent, with more than very low probability, the successful completion of the election. ...the government, even acting in collusion with a conspiracy of dishonest voters, cannot release a false tally without being detected by every honest voter, except with very low probability. "

In 1986, the paper *Distributing the Power of a Government to Enhance the Privacy of Voters*[BY86] demonstrates that it is possible to distribute the functionalities of government within an e-voting cryptographic voting scheme, such as that proposed in [CF85], so that it is possible

> "...to achieve privacy of individual votes in a much stronger sense without giving up any of the previously attained properties of robustness or veriability. This gives an electronic mechanism for holding a large-scale election amongst untrusted parties which is far more useful in real-world applications ..."

Work continued for a number of years on e-voting cryptography schemes. Many of the schemes went through a number of refinements but most were based on a small set of similar approaches (for example, Iversen proposes an election scheme based on e-payment protocols[Ive91]. A review of these is found in *A Practical Secret Voting Scheme for Large Scale Elections*[FOO93] where the authors also propose a new scheme which introduces new requirements concerned with maintaining privacy (if the administrators conspire with the vote counters) and ensuring fairness (so that no one can know an intermediate result of the voting). Their paper also formulated seven objectives of secure e-voting: completeness, soundness, privacy, unreusability, eligibility, fairness and verifiability. Later work suggests that these objectives could be better formulated and that they are incomplete; however, the

explicit statement of high-level objectives would become a standard component to later publications concerned with e-voting protocols.

By 1994, a number of schemes existed for verifiable secret elections. However, a common weakness was that they generated a receipt which could open up the possibility of a voter proving to others how they have voted. This in turn would compromise the fundamental requirement for coercion-free elections. Benaloh and Tuinstra address this issue in more detail in *Receipt-free Secret-ballot Elections*[BT94], where they demonstrate that it is not necessary to provide a receipt — showing how a voter has voted — in order to provide voter verifiability. This coercion-freeness is dependent on the physical assumption of private voting booths so that coercion is not possible during the process of recording a vote. In *Conducting secret ballot elections in computer networks: Problems and solutions*[NS94], Nurmi and Salomaa

> "...discuss problems related to devising a secret balloting system with the following properties: (1) all eligible voters and they only may vote, (2) all ballots are secret, i.e. do not reveal the identity of the voter, (3) all voters may check whether their ballots have been correctly assigned, (4) the voters may revise their ballots within a predetermined time, and (5) errors in ballot assignment can be corrected within a predetermined time."

A year later, Sako and Kilian published *Receipt-free Mix-type Voting Scheme*[SK95]. The interesting advance from the previous work is that they weaken the precondition for a physically isolated voting booth so that their scheme requires only a private channel for communicating messages between a central administration authority and the voters.

In 1995, Niemi and Renval propose, in *How to Prevent Buying of Votes in Computer Elections*, a scheme[25] for allowing a voter to verify that their vote has been counted correctly without being able to sell their vote without the coalition of all interested voting parties[NR95].

In 1996, Borrell and Rifa propose *An implementable secure voting scheme*[BR96] that:

> " ...reduces the cryptographic and communication requirements in comparison with other schemes which have been presented. No special comnunication channels are needed, and therefore it can be easily implemented on any existing computer network."

In their approach (using certification authorities to implement secret and authentic communications) they claim to "adopt simplicity as a crucial design requirement". However, as their high-level architecture involves nine components with a total of eighteen communication channels, it is not clear whether their proposal is as simple as they claim.

By 1996, there were many competing protocols for secure e-voting, yet it was not clear how these could be compared. Cranor addresses this problem (albeit indirectly) through an analysis of cryptographic techniques that could be used to ensure that e-voting systems exhibit fundamental characteristics: accuracy, democracy[26], privacy, verifiability, convenience, flexibility and mobility[Cra96]. This analysis demonstrates the importance of properly understanding and formulating requirements in order to judge the quality of the proposed systems in meeting the needs of different political systems in different democratic environments:

[25] This scheme reappears in 1999[NR99] where more technical details are provided and inefficiencies in the original protocol are removed.

[26] They rename the "democracy" characteristic as "invulnerability" in later work[CC97]

> "Although none of the voting protocols described here satisfy all of our desirable properties completely, some satisfy them well enough so as to be as good as or better than the traditional voting systems they may replace."

Hwang introduces a new requirement for "social acceptance" in *A conventional approach to secret balloting in computer networks*[Hwa96]. The scheme that is proposed is argued to be socially acceptable because:

> " . . . it resembles the conventional, non-computer, paper-based voting in basic procedures and organizational functioning."

In *A Simple Publicly Verifiable Secret Sharing Scheme and its Application to Electronic Voting*[Sch99], Schoenmaker presents a a new type of universally verifiable election scheme based on Publicly Verifiable Secret Sharing (PVSS). The scheme is more efficient than similar previous schemes[FOO93] and it focuses on the concept that anyone can verify that votes were correctly recorded and counted (not just the voters).

Schoenmakers then published *Compensating for a lack of transparency*[Sch00] where he reviews an actual system built on previous homomorphic encrypion schemes[CF85, BY86] and their implementation[CGS97].

In *A Verifiable Secret Shuffle and its Application to EVoting*[Nef01] provides a good overview of the problem of mixing votes and verifying the validity of the mix (through an audit):

> ". . . the problem of achieving the same kind of random, yet verifiable permutation of an input sequence is surprisingly difficult. The problem is that the data itself is either always visible to the auditor, or it isn't. If it is, then the correspondence between input records and output records is trivial to reconstruct by the auditor, or other observer. If it isn't, then input and output records must be different representations of the same underlying data. But if the output is different enough (that is, encrypted well enough) that the auditor cannot reconstruct the correspondence, then how can the auditor be sure that the shuffler did not change the underlying data in the process of shuffling? "

The paper then proposes an efficient (linear) method for solving this problem.

By 2003, many different schemes and protocols had demonstrated the feasibility of remote electronic voting. However, the reality was that any large-scale implementation of remote voting would most likely have to use the internet as its underlying communication architecture. Thus, the problem of unreliable and faulty communications arises. In *REVS — A Robust Electronic Voting System*[JZF03], Joaquim et al. propose a system specifically designed for distributed and faulty communication architectures (like the Internet). The key is that faulty, untrustworthy, communication should not compromise the protocols that are fundamental to the voting scheme. The paper provides a useful categorisation of voting protocols into three classes:

- **Blind signatures** are simple, with low computational costs, and are ballot independent (for example, see: [FOO93, CC97])
- **Mix-nets** require less voter's interactions, but need complex proofs of correctness (for example, see: [Cha81, SK95])
- **Homomorphic encryption** have complex mathematical structure and high computational costs. Further, these protocols do not work with many common types of ballots and tabulating processes (for example, see: [BY86, BT94, CGS97])

In *The design of a secure anonymous Internet voting system*[CJC04], yet another protocol scheme is proposed. An interesting aspect to this paper is that the authors attempt to carry out an objective evaluation of their proposed system against alternative systems. They formalise this analysis around ten criteria that had motivated more-or-less all previous published work on e-voting cryptography protocols:

- **Fairness —** No one can learn the voting outcome before the tally
- **Eligibility —** Only eligible voters are permitted to vote
- **Uniqueness —** No voter is able to vote more than once
- **Uncoercibility —** No voter can prove how he voted to others
- **Anonymity —** There is no way to derive a link between the voter's identity and the marked ballot.
- **Accuracy —** All valid votes are counted correctly and no vote can be altered, duplicated, or removed
- **Efficiency —** The computations can be performed within a reasonable amount of time
- **Robustness —** A malicious voter cannot frustrate or disturb the election
- **Mobility —** There are no restrictions on the location where voters can cast their ballots
- **Practicability —** No extra skills are required to vote and no additional equipment is required

From their analysis, it is clear that the engineering of e-voting systems involves compromise between these criteria. Their paper overlooks other criteria such as cost, quality of service (for example, how long it takes to vote), maintainability, verifiability, etc .... They suggest their proposed system is "very suitable for implementation on the Internet", yet they do not address the issue of denial-of-service attacks and they have a very weak interpretation of what it means for uncoercibility.

By 2004, the list of e-voting system security requirements, as reported in the literature, had been growing rapdily. Groth identified, in *Evaluating Security of Voting Schemes in the Universal Composability Framework*[Gro04a], that "designers of voting protocols face two problems: if they do not know the literature well they may miss a security requirement, and even if they do cover all known requirements this does not guarantee that new yet to be discovered requirements are satisfied by their voting scheme." To partially solve this problem, they suggest evaluating voting schemes in the universal composability (UC) framework of Canetti. They argue that their approach

> "...has the advantage that it covers many security requirements in a single security model. This simplifies security proofs since we only need to prove universal composability to prove all these specific security requirements. Our approach is also pro-active in the sense that using a general security model may mean that security requirements yet to be discovered are covered."

This paper is important because it demonstrates the utility of a standard model for requirements and their formal verification (albeit covering only a subset of the security criteria generally used for evaluating e-voting systems). We note that this work motivated the work reported, three years later, in *Simulation-based analysis of E2E voting systems*[dMPQ07], where the authors investigate the use of techniques from the simulation-based security tradition (in the line of the UC framework) for the analysis of voting protocols.

In *The Vector-Ballot E-Voting Approach*[KY04] Kias and Yung argue that since each of the three basic types of e-voting scheme (mix nets, homomorphic encryption and blind signatures) have their own strengths and weaknesses, a better approach would be to try and combine/compose these schemes. They argue that their proposed combination of "mix networks and homomorphic encryption under a single user interface" results in a system where the three basic properties of "(i) efficient tallying, (ii) uni- versal verifiability, and (iii) allowing write-in

ballot capability (in addi- tion to predetermined candidates)" are better addressed than with any of the basic types of scheme.

The critism that homomorphic encryption schemes are resource intensive (and inefficient) is addressed in *Multiplicative Homomorphic E-Voting*[PAB+04], where the authors demonstrate that a multiplicative scheme is more efficient than the previously published additive homomorphic schemes. They also demonstrate that their scheme is more efficient than mix net voting schemes when the number of candidates is small.

In *Secret-Ballot Receipts: True Voter-Verifiable Elections*[Cha04] Chaum proposes using receipts that are encoded visually with two layers so that the voter can, using one layer, verify that their vote is counted without being able to demonstrate how they have voted; except to an entity who has a copy of the 2nd layer. This scheme requires sophisticated printing processes and add complexity to the interaction between voters and the voting system. This scheme is improved upon in *A Practical, Voter-Verifiable Election Scheme*[CRS05] where the *Prêt à Voter* scheme replaces the visual with a more conventional representation of the vote: "ballot forms with the candidates or voting options listed in one column, and the voter choices entered in an adjacent column." The paper identifies that the scheme requires further work and analysis of risks but it clearly offers many advantages over previous schemes. We note that a similar layered approach is seen in the *Scratch and Vote* protocol[AR06] but it uses the more familiar technology of scratch surfaces to simplify the vote and audit process.

In *Analysis of an Electronic Voting Protocol in the Applied Pi Calculus*[KR05] Kremer and Ryan demonstrate that one can formally verify properties of cryptographic protocols in an automated fashion, using a protocol verification tool. They also show that the tool is unable to prove all properties automatically and that a manual proof must be used and checked in such a case.

In *Coercion-Resistant Electronic Elections*[JCJ05] the problem of coercion resistance with remote voting is readdressed. The clear issue is that a coercer may be present when a voter records their vote and a scheme for e-voting must provide a means for ensuring that in such a circumstance the coercer cannot be sure that the vote recorded (under their supervision) is the one that will actually be counted. The first contribution of this paper is to describe and characterize "a new and strengthened adversary for coercion who may demand of coerced voters that they vote in a particular manner, abstain from voting, or even disclose their secret keys". The second contribution is to define a scheme that is "coercion-resistant as it is infeasible for the adversary to determine whether a coerced voter complies with the demands". This scheme is based on the use of voter credentials and an underlying architecture for producing real (and false) credentials and for checking credentials in a way that is guaranteed to be hidden from a potential coercer. The authors note that the scheme is not yet practical because of the overhead for tallying authorities as the number of voters grows.

In *Cryptographic Voting Protocols: A Systems Perspective*[KSW05] the authors identify several potential weaknesses in cryptographic voting protocols which became apparent only when attacks are considered in the context of the entire voting system:

> "These attacks could compromise election integrity, erode voter privacy, and enable vote coercion. Whether our attacks succeed or not will depend on how these ambiguities are resolved in a full implementation of a voting system, but we expect that a well designed implementation and deployment may be able to mitigate or even eliminate the impact of these weaknesses. However, these protocols must be analyzed in the context of a complete specification of the system and surrounding procedures before they are deployed in any large-scale public election."

The main contribution of this paper is to show the importance of modelling a voting system as a whole and that the

security of a system cannot be guaranteed throw analysis of the underlying cryptographic voting scheme alone. We note that this paper motivated Ryan and Peacock to revisit the *Prêt à Voter* scheme from a system perspective[RP05] and they demonstrate that the scheme is "remarkably robust to most of the vulnerabilities" that can arise from a system view.

In *E-voting: Dependability Requirements and Design for Dependability*[BLRS06] the authors illustrate the fault-tolerant design implied by the *Prêt à Voter* scheme and discuss "the allocation of dependability requirements to subsystems for defence against both accidental fault and malicious attacks". A main contribution of the paper is the discussion concerning the balance between design time and run time measures for achieving and assessing the dependability of complex voting systems. They highlight the need for considering the whole socio-technical system, and for integrating security and fault tolerance viewpoints:

> "Our discussion has emphasised three aspects: requirement specification, with the need to translate society's informal requirements and to consider how threat profiles change compared to those affecting non-electronic elections; design issues for the fault-tolerant system, including the need to complement the basic cryptography-based ideas with explicit methods for assurance of the detection mechanisms and explicit recovery mechanisms, subsystem-level dependability requirements and the concern for denial-of-service attacks; and the need to integrate technical considerations with psychological and social ones that determine the threat profile, the voters' reactions and the effectiveness of the socio-technical mechanisms for error detection and recovery."

In *Coercion-Resistance and Receipt-Freeness in Electronic Voting*[DKR06], Delaune, Kremer and Ryan examine the relation between privacy, coercion-resistance and receipt-freeness as in much of the previously published literature the terms are not used clearly. They formalise the following definitions in the applied pi calculus framework:

- **Receipt-freeness —** a voter does not gain any information (a receipt) which can be used to prove to a coercer that she voted in a certain way. This is quite naturally formalised as an observational equivalence.
- **Coercion-resistance —** a voter cannot cooperate with a coercer to prove to him that she voted in a certain way. In this case, observational equivalence is not flexible enough, and they generalise it to a notion which they call adaptive simulation.
- **Privacy —** the system cannot reveal how a particular voter voted (also formalised as an observational equivalence).

They then go on to "prove that, in accordance with intuition, coercion-resistance implies receipt-freeness, which in turn implies privacy". They note that Jules, Catalano and Jakobsson also provide formal definitions of these concepts[JCJ05] but that a comparison is difficult because of large differences between the underlying models.

In Kleptographic Attacks on E-Voting Schemes[GKK$^+$06] the authors identify that evoting systems based on particular cryptographic schemes may be vulnerable to attacks using kleptograhic techniques: "using randomness in e-voting schemes yields a threat of constructing a subliminal channel by a malicious voting machine". They show how information leaked by specific types of kleptographic attack can be "retrieved only by a party possessing a certain secret key and that such a malicious implementation neither changes the protocol executed nor can be detected without reverse engineering of the software running on the device". Thus, if one cannot trust the machine manufacturers then there is a potential for them to include a kleptographic trapdoor in their products which can be found only through analysis and verification of all system components.

In *A framework and taxonomy for comparison of electronic voting schemes*[SP06] the authors propose three different classes of requirements:

- **General Security Requirements —** Eligibility, Privacy, Verifiablity, Dispute-freeness, Accuracy, Long-term Privacy and Fairness
- **Adversary Counter-attack Requirements —** Robustness, Receipt-freeness and Incoercibility
- **System Implementation Requirements —** Scaleability and Practicality

The main contribution of the paper is a detailed analysis of more than 20 schemes against these requirements. A weakness is that they do not try to match their requirements against similar lists provided in previous publications(for example, in [CJC04]).

In *Ballot Casting Assurance via Voter-Initiated Poll Station Auditing*[Ben07] Benaloh argues that "if done properly, substantial integrity can be obtained by giving voters and observers the option to challenge ballot validity without requiring all voters to do so." Hence, the majority of voters do not need to follow an elaborate process of vote verification but they can be reasonably sure that a high-quality audit is carried out provided a small minority of voters do carry out some verification. The paper proposes a generic voting process that is suitable for any form of direct ballot casting scheme (for example, *Prêt à Voter*[BLRS06] and *ThreeBallot Voting*[RS07]). They note that direct ballot casting is simple but fragile and that the process must be carefully managed. It then describes a detailed process that mitigates all known threats which provides a "blueprint for how verifiable, open-audit elections can reasonably be conducted in practice". The paper's main weakness is that the threats and process are not formally modelled, and consequently the reasoning, concerning properties of the process whilst under attack, lack rigour.

The notion of bare-handed voting, where the voter requires no computational power, was first introduced by Chaum[Cha04] but it has the disadvantage that the voter must reveal their vote to the voting booth (machine). Thus, a voter has to trust that the software (algorithm) running on the machine is the correct one. In *Practical high certainty intent verification for encrypted votes*[RTS07] this problem of trust is addressed by allowing the voter the use of a computer device but only at a pre-processing stage — the voting itself is done bare-handedly. Thus, the voter maintains his privacy with respect to the voting booth and withithout sacrificing secrecy to some other party (like the ballot generators). This is done by involving the voter (with the aid of a computing device) in the ballot generation process. The authors note that the protocol proposed is a variation on the *Prêt à Voter*[CRS05] and *Scratch and vote*[AR06] schemes. One problem that exists with this protocol (and the schemes on which it is based) is that it can not settle disputes when auditing finds inconsistency between voting data: the protocol supplies no way of determining whether the voter is honest and the booth is dishonest or vice versa. However, the authors suggest that this problem is easily solved in practice.

In *Three Voting Protocols: ThreeBallot, VAV, and Twin*[RS07], Rivest and Smith propose the goal of achieving the same security properties as recently proposed cryptographic voting protocols using only paper ballots and no cryptography. They then introduce three different protocols that go some way to meeting this goal:

- **ThreeBallot —**

  > ... "each voter casts three paper ballots, with certain restrictions on how they may be filled out. ... A voter receives a copy of one of her ballots as her "receipt", which she may take home. Only the voter knows which ballot she copied for her receipt. The voter is unable to use her receipt to prove how she voted or to sell her vote, as the receipt doesn't reveal how she voted. A voter can

check that the web site contains a ballot matching her receipt. Deletion or modification of ballots is thus detectable; so the integrity of the election is verifiable."

- **VAV —**

  VAV is like ThreeBallot, except that the ballotmarking rules are different: one ballot may "cancel" another (VAV = Vote/Anti-Vote/Vote).

- **Twin —**

  "...is based almost entirely on Floating Receipts[27] where each voter casts a single ballot and takes home a single [floating] receipt."

It remains to be seen whether these simple voting procedures will withstand more rigorous analysis by academics and whether they can be adopted in practice. They provide clear mechanisms for detecting problems but this raises issues of how to deal with such problems if/when they arise. We note that the authors do not file any patents on these approaches; and they encourage others who work on extensions, improvements and variations to act similarly.

Auditing is a key requirement for all voting schemes. In *Casting Votes in the Auditorium*[SW07] Sandler and Wallach propose a secure auditing infrastructure — the *Auditorium* — to provide a verifiable, global record of critical election events; "where each event is irrevocably tied to the originating machine by a digital signature, and to earlier events from other machines via hash chaining".

In *Mobile implementation and formal verification of an e-voting system*[CFM+08] the authors propose a mobile implementation (using Java MIDlets) of a variation of the Sensus e-voting protocol[CC97]. A major weakness of this protocol is that it allows one of the entities involved in the election process to cast illegitimate votes for registered voters who have abstained from voting. To address this problem they adapt the protocol and use a CCS-like process algebra to model the system (including the new protocol) and a model checker to verify that this weakness no longer exists. The approach in the paper is noteworthy: they use formal methods to demonstrate a weakness in an existing protocol and to develop a variation of the protocol in which the weakness is shown to be removed in a formal, automated, fashion using model checking tools. One issue that arises, but which is not addressed in the paper, is whether previously verified (proven) properties of the original protocol are compromised by the changes made in producing the new version.

In the 2008 presidential elections, the use of optical scanning equipment was widespread. The need for higher levels of integrity in these systems is addressed in *Scantegrity II: End-to-End Verifiability for Optical Scan Election Systems*[CEC+08], where the authors propose a novel use of confirmation codes printed on ballots in invisible ink. The paper reports that the system "...has been implemented in open-source Java with off-the shelf printing equipment and has been tested in a small election." The use of invisible ink allows correctness of the receipt to be verified by giving a voter a choice to audit it after it is created but before it is seen, "proving to the voter that (with high probability) the receipt was generated correctly." (This idea is also seen in the *Scratch and Vote* protocol[AR06].) It remains to be seen whether the scheme fulfils its "promise of being the first end-to-end voting system to come into use in public sector elections".

The well-known problem of the "Italian attack" is addressed in *Coercion-Resistant tallying for STV voting*[TRN08]. This attack is unique to STV schemes where votes are tallied in public: they suffer from a coercion problem when there are many candidates, because a coercer can demand a certain permutation from a voter and then check

[27] With floating receipts, voters may take home a copy of another (unknown) voter's ballot.

whether that permutation appears during tallying. The paper shows that this attack can be addressed through the verifiable tallying of encrypted STV votes. However, they acknowledge that their proposal requires further analysis and refinement.

The problem of anonymity in STV voting is also addressed in Analysis, Improvement and Simplification of Prêt à Voter with Paillier Encryption[XSHT08] which more generally examines the problem of leakage of voter's choice information in different election types. (Details of the STV analysis is found in *Implementing STV securely in Prêt á Voter*[Hea07].)

In *Verifiable Anonymous Vote Submission*[ZA08] adapt the REVS architecture[JZF03] to fulfil requirements of anonymity and verifiability. One interesting aspect of the proposed scheme is that there is an additional requirement that "it should be possible to submit the same vote several times and to detect and drop vote replicas at the tallying phase".

4.4.2 Software Product Lines

Although the origins of SPLs are decades old[McI68, Par76], most of the initial research was carried out in the late 1990's[Wit96, AW97, BFK+99, Bos99b, Bos99a, DS99]

At the turn of the century, the field expanded. We focus on the three main sub-topics which are most directly relevant to this proposal: features and interactions, the need for formality, and industrial case studies.

4.4.2.1 Features and Interactions

A breakthrough paper that linked features and variability was published in 2001 by van Gurp, Bosch and Svahnberg[vGBS01]:

> " This is an important characteristic of our approach as it is an important means for early identification (i.e. before architecture design) of variability needs in the future system. "

In 2002, we saw the emergence of a Feature-Oriented Domain Analysis (FODA) approach to SPLs[KLD02]:

> "The Feature-Oriented Reuse Method concentrates on analyzing and modeling a product line's commonalities and differences in terms of features and uses this analysis to develop architectures and components. The FORM explores analysis and design issues from a marketing perspective."

In 2003, Kuloor and Eberlein identified cross-cutting concerns which could be viewed as non-local features. As a solution they proposed: *Aspect-Oriented Requirements Engineering for Software Product Lines*[KE03].

Many more advances were made in modelling SPL requirements using features, but it was clear that integration (and interactions) were a major problem[Dhu06]. In particular, this paper highlights the common problem of verification techniques, which were (and still are) an open fundamental research question:

> "We've built some initial models of our industry partner's product line to test the concepts developed so far. Various other iterations will be necessary to cover all necessary aspects to model a real world situation. We still haven't devised any mechanisms to verify the consistence and completeness of the model automatically. Completeness of the instantiated products also needs to be confirmed. By and large, an integrated model consisting of features and architectural elements can to some extent support and automate various activities of product line engineering."

In 2007, Decker and Dager take a customer oriented view of SPLs[DD07] They state — in *Software Product Lines Beyond Software Development* — the importance of customers understanding feature-oriented requirements:

"This requires them to understand the various features and capabilities of the electronic system at least at a high level."

4.4.2.2 Formal Methods and Modelling Critical Functionality

In 2003, there was innovative research into using UML to add rigour to the modelling of SPLs[ZHJ03]. They propose:

> "a new approach for modeling product lines with UML integrating functional, static and dynamic aspects. First, an extension to use cases, class diagrams and sequence diagrams is proposed. The coherence between parts of the product line models is then ensured by OCL (Object Constraint Language) constraints."

In 2007, the paper by Liu [Liu07] specifically examines the use of SPLs in safety critical system development:

> "Safety-related feature interactions are central to the development of safety-critical, software product lines, with the challenge of balancing safety assurance and reuse management."

A year later, in *A Requirements-Based Taxonomy of Software Product Line Evolution*[SE08] the authors identify the need for modelling SPLs so that they can be better evolved and maintained. They identify a major problem:

> "So far most work on product line evolution has focused on specific approaches to supporting special cases of the evolution problem."

A more formal modelling of SPLs is the only way to address this issue.

In *SAT-Based Analysis of Feature Models is Easy*[MWC09] the authors observe that: "feature interacton analyses essentially reduce to satisfiability problems; thus, it suffices to study this more abstract problem to obtain conclusions about the analyses." Based on experimentation, they claim that:

> "Unlike with the general SAT instances, which fall into easy and hard classes, the instances induced by feature modeling are easy throughout the spectrum of realistic models. In particular, the phenomenon of phase transition is not observed for realistic feature models."

It remains to be seen whether this result is generalisable and scaleable; but it does further motivate the use of formal modelling techniques for the development of SPLs.

More recent work has highlighted the problem of reasoning about interactions and liveness properties[CHS+10]. Although this paper proposes model-checking techniques, it is clear how closely it complements previous work on *Composing Fair Objects*[HGM00] from almost a decade past.

4.4.2.3 Industrial Case Studies

The first large-scale review of SPLs in industry was reported by Bosch in 1999[Bos99b]. *Product-line architectures in industry: a case study* concludes that:

> "Product-line architectures can and are successfully applied in small- and medium-sized enterprises. These organizations are struggling with a number of difficult problems and challenging issues, but the general consensus is that a product-line architecture approach is beneficial, if not crucial, for the continued success of the interviewed organizations.

”

There followed a number of reports of the successful development of SPLs in specific industries. A good example is the *Component-based product line development of Avionics Software*[Sha00] where “validation of the product line approach led to broader application of the technique.”

Further growth — over a number of years — of the uptake of SPLs in industry led to increased attention in the general software engineering community[EBB06]. In this paper — *Software product line modeling made practical* — the authors “describe an approach integrating use case modeling and feature modeling to support the description and maintenance of a common and complete use case model for an entire family of systems.” Their approach, referred to as PLUSS (Product Line Use case modeling for Systems and Software engineering), is illustrated within the context of a number of large-scale industrial projects.

4.4.3 Event-B: Mixed Models Research

Our proposed research approach is to use the Event-B language and the RODIN tool[Abr07, ABHV06, AH07] for specification and verification of our e-voting product line models. We have already stated the need for different modelling languages; and fortunately there has been much recent work on extending/integrating Event-B with other formalisms.

non-atomic operations and features for specifying implementation level details”.

The integration of UML and B has been reported in *UML-B: A Plug-in for the Event-B Tool Set*[SB08], where they state:

> “UML-B provides tool support, including drawing tools and a translator to generate Event-B models. When a UML-B drawing is saved the translator automatically generates the corresponding Event-B model. The Event-B verification tools (syntax checker and prover) then run automatically providing an immediate display of problems which are indicated on the relevant UML-B diagram.”

This work was extended in [SBS09], where a useful case study demonstrates the method for integrating the different models and views in a coherent fashion. Similar work was proposed in *Using UML Activity Diagrams and Event-B for Distributed and Parallel Applications*[YA07].

In 2008, Edmunds and Butler report on *Linking Event-B and Concurrent Object-Oriented Programs*[EB08] They “show how Event-B models can be linked to concurrent, object-oriented implementations using an intermediate, object-oriented style specification notation.” They also “automated the translation process with an Eclipse plug-in which produces an Event-B model and Java code”. To conclude, they “build on techniques introduced in UML-B to model object-oriented developments”. (This follows the previous work that was reported on integrating CSP with B[BL05].)

We finish this review of related work (in Event-B) by noting that Poppleton has already published research into modelling of features using Event-B[Pop07, PFF+08] and the problem of composition[Pop08]. This work suggests that a mixed-model approach could be beneficial.

4.4.4 Education Research

4.4.3.1 Teaching Formal Methods

The need for students to be able to use general software development tools is widely accepted by industry; but the importance of them being able to use formal methods tools is not. One used to be able to argue that formal

methods were not used in industry because they were not mature enough — and therefore it would be difficult to motivate students to learn how to use them[GM98] — but this is no longer the case.

In 2000, Jeanette Wing wrote about weaving formal methods[Win00]:

> "Rather than treat formal methods solely as a separate subject to study, we should weave their use into the existing infrastructure of an undergraduate computer science curriculum. In so doing, we would be teaching formal methods alongside other mathematical, scientific, and engineering methods already taught. Formal methods would simply be additional weapons in a computer scientist's arsenal of ways to think when attacking and solving problems.
>
> My ideal is to get to the point where computer scientists use formal methods without even thinking about it. Just as we use simple mathematics in our daily life, computer scientists would use formal methods routinely."

She then goes on to identify the common core elements that need to be taught: state machines, invariants, abstract mappings, composition, specification, induction and verification. She states that tools are critical: model checkers, specification checkers and theorem provers.

Students learn that nondeterminism is a very powerful mechanism. We first noticed this when we analysed how best to teach formal specification as part of requirements engineering[Gib00].

Parnas and Soltys address the need for a "Basic Science for Software Developers"[PS06], stating:

> "The fundamental properties of computers are very important because they affect what we can and cannot do. Sometimes, an understanding of these properties is necessary to find the best solution to a problem. In most cases, those who understand computing fundamentals can anticipate problems and adjust their goals so that they can get the real job done. Those who do not understand these limitations, may waste their time attempting something impossible or, even worse, produce a product with poorly understood or not clearly stated capabilities. Further, those that understand the fundamental limitations are better equipped to clearly state the capabilities and limitations of a product that they produce. Finally, an understanding of these limitations, and the way that they are proved, often reveals practical solutions to practical problems. Consequently, "basic science" should be a required component of any accredited Software Engineering program."

Habrias has written about the problems of teaching formal methods when the students do not have a good understanding of foundational mathematics such as logic and set theory[Hab08]. Much of the literature on teaching formal methods directly addresses the need for firm mathematical foundations. We believe that the problem-based learning approach helps students with the mathematics because they learn the mathematical concepts in the context of their practical application.

More recently, Kiniry and Zimmerman discuss the use of "secret ninja" techniques[KZ08] "to integrate applied formal methods into software engineering courses." They demonstrate that formal methods can be taught through "stealth" (without calling them formal methods) in a number of different courses; but note that this success would not have been possible without good tool suppport. Their work is founded mostly on applying the design-by-contract paradigm. This demonstrates that formal methods can and should be used in the teaching of software design; and this view is supported by other research[GLR08b].

4.4.3.2 Education Theory

There are numerous complementary, and competing, theories of learning. The review by Hilgard and Bower published over half a century ago[HB56] is a good introduction to the foundations of learning theory. In this proposal, we review the work of the researchers that have had most influence on our own research into teaching formal methods.

Cognitive structure is the concept central to Piaget's theory. (See the work by Brainerd[Bra78] for a good overview and analysis of Piaget's seminal contribution.) These structures are used to identify patterns underlying certain acts of intelligence, and Paiget proposes that these correspond to stages of child development. Piaget's most interesting experiments focused on the development of mathematical and logical concepts. However, his work predates the development of software engineering as a discipline.

Piaget's theory is similar to other *constructivist* perspectives of learning (e.g., Bruner [Bru66]), which model learning as an active process where learners construct new concepts upon their current knowledge and previous experience. As a result of following this theory, teachers encourage students to discover principles by themselves: this is the foundation upon which problem-based learning is built.

Similarites can be seen between the constructivist view and the *theories of intelligence* such as proposed by Guildford's *structure of intellect* (SI) theory [Gui67] and Gardner's *multiple intelligences*[Gar83]. Typically, these theories structure the learning space in terms of practical problem solving skills.

Piaget's ideas also influenced the seminal work by Seymour Papert in the specific domain of computers and education[PS80]. Papert argues that children can understand concepts best when they are able to explain them algorithmicaly through writing computer programs.

We were also influenced by the domain of teaching mathematics. In particular, Alan Schoenfeld argues that understanding and teaching mathematics should be treated as problem-solving [Sch85]. He identifies four skills that are needed to be successful in mathematics: proposition and procedural knowledge, strategies and techniques for problem resolution, decisions about when and what knoweldge and strategies to use, and a logical *world view* that motivates an individual's approach to solving a particular problem.

To conclude our review we mention Blooms taxonomy[BEF$^+$56] of educational objectives which is a fundamental model of learning, providing a well-accepted foundation for research and development into the preparation of learning evaluation materials. It structures understanding into 6 distinct levels: Knowledge, Comprehension, Application, Analysis, Synthesis and Evaluation.

4.4.3.3 Problem-Based Learning (PBL) in Computer Science and Software Engineering (CSSE)

While there is no universal definition of PBL we present definitions from the last three decades. PBL was defined by Barrows and Tamblyn[BT80] as "the learning which results from the process of working towards the understanding of, or resolution of, a problem. The problem is encountered first in the learning process". Woods defined it[Woo96] as "an approach to learning that uses a problem to drive the learning rather than a lecture with subject matter which is taught." Torp and Sage define it[TS02] as "Focused, experiential learning (minds-on, hands-on) organised around the investigation and resolution of messy, real-world problems."

In 2003, Curran discussed the balancing required between Computer Science (CS) and Software Education (SE) education[Cur03]:

> "It is no longer clear whether SE topics reflect current industry needs or whether they are intended to lead and update industry practices. But regardless of who leads whom, without some sort of rapid, two-way communication, we run the risk of producing graduates who are out of touch, require much re-training, and have trouble competing. Industry might indicate that they need specific skills and

> knowledge from their CS employees, and that the special skills required of software engineers would be performed by software engineers, not by CS majors."

He concluded by stating:

> "...individual departmental goals for a degree in CS and the role of SE in the curriculum should be clearly understood so that a balance can be struck between academic topics and skills training."

We believe that PBL offers a natural solution to achieving this required balance and in using formal methods to bridge the gap between CS and SE.

One of the major obstacles to the implementation of PBL, within any discipline, is the lack of a good set of problems. However, good PBL problems usually do not appear in textbooks[TCL05]. Clearing houses provide an avenue to allow for the sharing of problems, but unfortunately there is a lack of CS&SE problems[28].

In practice, it is very difficult, if not impossible, to fairly evaluate whether the objective of improving the students' software development skills is being met by our PBL approach to teaching formal methods. O'Kelly and Gibson have discussed the issues that arise when trying to validate PBL in the context of teaching programming[OG05], and many of the issues that they identify are relevant when analysing whether the formal methods problems are teaching the students how to be better software engineers (i.e. engineer better software)

In 2006 Wing discusses the importance of computational thinking in education[Win06]:

> "Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking includes a range of mental tools that reflect the breadth of the field of computer science."

She then goes on to discuss the characteristics of such thinking:

1. Conceptualizing, not programming;
2. Fundamental, not rote skill;
3. A way that humans, not computers, think;
4. Complements and combines mathematical and engineering thinking;
5. Ideas, not artifacts;
6. For everyone, everywhere.

Such computational thinking starts from a very early age[GO05] and should be exploited in the teaching of computer science (and formal methods) in schools[Gib08a]. Our experience shows that looking at simple formal methods problems with school children improves their ability to think computationally. Thus, we believe that this should also be true for university students.

4.4.3.4 E-voting: an educational case study

In 2005, Armen and Morelli wrote about *Teaching about the risks of electronic voting technology*[AM05]. They note that:

[28] There is an abundance of CS programming problems available; however, the vast majority of these problems place emphasis on the learning of a particular programming concept rather than problem solving.

> "computer scientists are increasingly called upon to help concerned citizens understand the risks involved in the current generation of electronic voting machines.. . . "

They go on to "suggest ways that discussions of the risks and the attendant societal and ethical issues might be incorporated into the computer science curriculum." They do not suggest that the problem be used in any particular module/course, but argue that it could fit in many places in the curriculum.

More recently, Bishop and Frincke report on *Achieving Learning Objectives through E-Voting Case Studies*[BF07]. They focus on teaching security:

> "the rapidly increasing use of electronic voting machines in US elections provides a wonderful opportunity to teach students about computer security. The complexity of transitioning from traditional voting to an electronic environment allows educators to highlight threat models, requirements, and trade-offs involving e-voting in the context of ongoing international discussions and current events. Few issues include such a wide range of considerationsŮfrom the competing demands of accessibility and confidentiality to threat models incorporating coercibility and vote selling and even the business element of whether funding high assurance is a good way to increase voter confidence. "

This paper follows a PBL approachn, based on explicit identificiation of learning objectives. The paper also emphaises the role of formal specification/modelling.

In our own research on *Weaving a Formal Methods Education With Problem-Based Learning*[Gib08b] we identify the value in problems such as e-voting which can be used in most, if not all, of the modules that one is likely to have in a CS/CSSE programme:

> "The problem has been re-used in teaching the following modules: introduction to programming, object oriented programming, data structures and algorithms, HCI, testing, requirements and design, rigorous software process, software process improvement."

4.5 Research Method: concrete to abstract and back to concrete

The research method is based on the following initial engineering programme:

- Reverse engineer our existing e-voting system software (from the SAVE project) into a set of re-usable artefacts (classes, interfaces and design structures)
- Construct the software for a new e-voting system, trying to maximise re-use of the components from the first step.
- Analyse where formal concepts such as invariants, preconditions and postconditions can help in the testing of the systems produced.
- Formally model the common aspects between both systems (using Event-B) and experiment with compositional verification techniques.
- Build a prototype of a small e-voting system SPL, based on the previous Event-B models and proofs.
- Using this prototype, construct a novel voting system, following a correct-by-construction approach based on refinement.
- Examine how the SPL itself can be extended and refined, linking regression testing of our concrete implementations with re-usable proof in our abstract specifications.

- Construct a family of formal requirements models for a range of e-voting systems where common properties have been verified through a single verification of the SPL formal model from which they were derived.
- Choose one such instance and develop it using three different approaches — formally using the formal SPL, informally using the implemented SPL components (but not the formal Event-B models), and without using a SPL of any kind. Compare and contrast the quality of the final systems produced in this way.

We see that this approach can be summarised as going from concrete to abstract and back to concrete. Applied research will feedback into foundational research, whose results will then feed forward into real-word development. A critical part of this research is that we will not restrict ourselves to a single semantic view on the problem of interactions in product lines. There is much current work on integrating Event-B with other modelling languages and tools, and we envisage adopting a mixed-semantic approach around the RODIN tool and associated plug-ins.

As a final component of our research, we also incorporate an educational research plan, which will leverage and test the results of the engineering research:

- Develop a postgraduate module on model driven development (MDD) and SPLs using the e-voting SPL as our standard problem.
- Incorporate the e-voting SPL problem into a postgraduate module on formal methods.
- Construct a PBL project to test whether students can build their own specific e-voting system using a formal SPL, and analyse the verification techniques.
- Integrate the e-voting SPL problem into other complementary modules such as design (with UML), testing and software maintenance.

4.6 Fundamental Research: Feature Interactions in SPLs

An e-voting system has a myriad of layers of inter-related legal requirements to meet. Further, each voting system has to meet specific needs which are not directly addressed by the laws and standards. The requirements of the system must somehow integrate these specific needs with multiple layers of laws and standards. As changes are made to requirements within different layers, in parallel, then who is responsible for ensuring that the requirements can be re-integrated in a coherent manner?

As an example, consider the most challenging requirements integration problem in e-voting: how to ensure both anonymity and verifiability? This the classic example of requirements that appear to be contradictory — how can an elector's ballot be kept secret when we wish the elector to be able to verify that it has been correctly counted? It would appear that verifiability requires a voter to be able to follow their ballot through the count process (at the very least) but how can they do that without signing it, and if they sign it then how can it be kept anonymous? Recent research in cryptographic e-voting protocols[RS07, ZA08, XSHT08] suggests that these two requirements can be met, provided we refine the notion of verifiability and we require the electors to follow specific verification procedures. However, it is clear that there are a number of subtle interactions between anonymity and verifiability when they integrate with other voting features, which results in different protocols making different compromises between competing criteria[CJC04]. Another major problem is that many of these schemes depend on a reliable non-local network during the time in which an elector records their individual vote and so the *QoS* feature is compromised.

Such interactions are common to all complex systems; and must be addressed in any proposed development of a software product line for all domains, and not just e-voting.

4.7 Applied Research: A SPL for E-voting

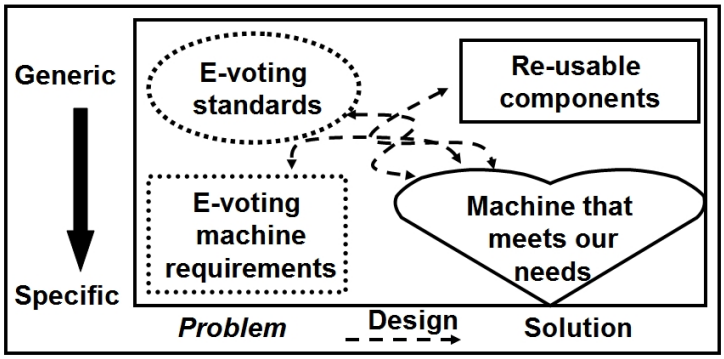

Figure 25: The Re-use Knot

In figure 25, we see a graphical representation of our main problem, that of re-use: we know that requirements modelling is critical in the production of quality systems, we understand that our system must meet certain standards and we wish to re-use particular components in order to provide innovative functionality. In effect, there is a complex knot of lines of re-use that must be managed during design.

In figure 26, we illustrate how a high-level object-oriented view of the requirements of a particular voting system, including optional features, can aid the process of identifying how features can be implemented using re-usable components, and how these features can be composed in order for a voting system to meet its requirements. The model in figure 26 is not intended to be generic; it is an example of how a requirements model for a specific voting system can be engineered re-using a core architecture, together with feature increments.

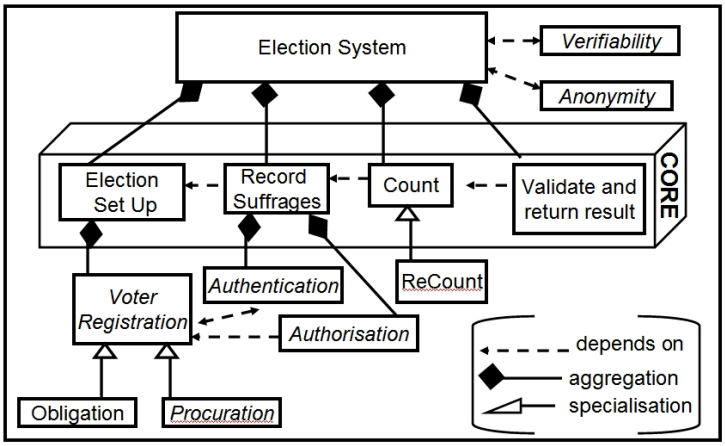

Figure 26: Adding Features To Architecture

In the diagram we see that we have introduced elector authentication and authorisation features. We do not categorise these as core features because a minority of elections do not have these requirements. In the figure we show

the feature composition for authentication, authorisation and registration as simple aggregation relationships with a single core feature. In the model we also see features that are introduced as specialisations of existing features: procuration and obligation are specialisations of the registration process, whilst we chose to model recounting as a specialisation of the count feature. Of course, each of these optional features are open to specialisation themselves.

Two other optional features of interest are voter anonymity and verifiability. It is not clear, due perhaps to our lack of domain understanding, how such features could be cleanly plugged into a core architecture. In our model in figure 26 we have represented these features as being interdependent with the whole voting system. In the next subsection we show how anonymity is a much more complex feature, with many variants, than one would initially suspect. We believe that aspects may hold the key to modelling e-voting features, like anonymity, that cut across multiple components of the architectural core.

Through initial analysis of the standards — both national and international — we identified that some of the chosen system requirements were not addressed, some were partially addressed, and some were completely addressed. Also, many of the standards were not applicable to the chosen system. This can be seen in the top left of figure 27.

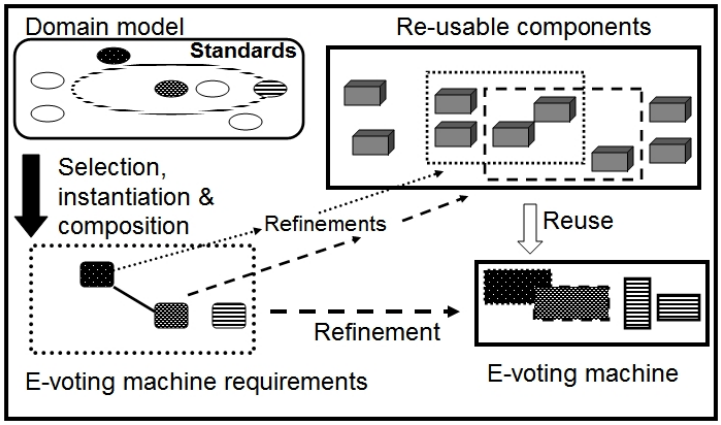

Figure 27: Domain Modelling and Standards

In order to move from generic standards to specific requirements we need a complex process of choosing the standards that are applicable, instantiating them to the parameters of our chosen system and then composing them with the requirements that are specific to our chosen system. This is represented in the left hand side of figure 27. We understand certain of our requirements well-enough that we can start to map them to particular re-usable components. This can be seen in the top right of figure 27.

We have decided that our domain model should not be structured around the poorly engineered standards documents. We will — where possible — try to validate our domain model against current standards, but the domain model now plays a more important new role: as a requirements specification for our software product line. The goal is to have a direct mapping between components in the SPL and the features (requirements increments) in the domain model. This is illustrated in figure 28.

There are obvious advantages with this framework when compared with that in figure 27. Firstly, verification of the correctness of refinements — a good example is the refinement of the voter interface[CGM07b] — can be done

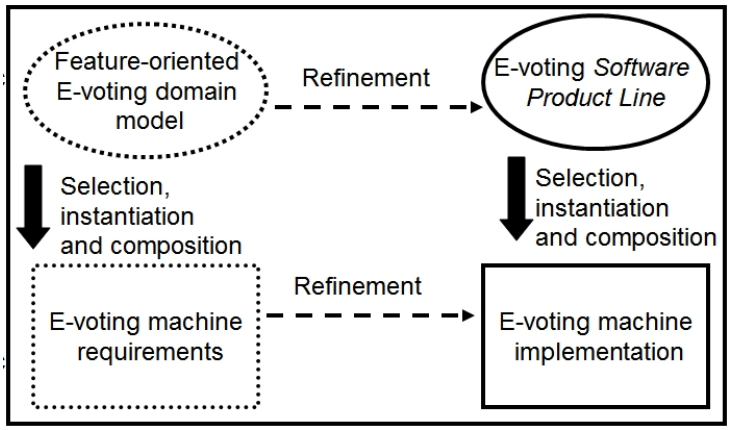

Figure 28: The E-voting Feature-oriented SPL

at the generic level (facilitating re-use). Secondly, new components and features can be soundly integrated into the framework through the process of abstraction into the domain model. Thirdly, we should be able to formalise different composition mechanisms in order to automate the configuration process.

A major problem with e-voting systems is that they need to be trustworthy and trusted [Gib07]. There are two key properties of SPLs that our proposal will examine: late binding and openness. Expecting electors to trust a voting machine has, until now, required them to trust that they have been properly verified to do what they are supposed to do. Thus, they trust the machines indirectly through their trust of the agents who have verified them. This trust is dependent on the electors knowing that the machines they use to record their individual vote are precisley the machines that have been verified. In a SPL, late binding of components opens up the question of whether the precise system in front of the users has ever been verified. Furthermore, leaving a variant open to later specialisation begs the question of whether trust can be compositional: if the delivered system is trusted and the specialised variant is trusted then can the new system that binds the new variant to a specific feature be trusted without having to reverify the whole system? Expecting users to trust the system in this way is unreasonable if we cannot guarantee that our verification mechanisms are compositional. This returns us to developing an SPL specific to voting systems that uses composition mechanisms that can guarantee absence of unwanted feature interactions (before the system is delivered).

4.8 Educational Research: Teaching Formal Software Engineering

The final part of our proposed research focuses on the educational value of using this research in the classroom.

It is a universal challenge to bridge the gap between academia and industry, and between theory and practice. This challenge is particularly critical in the discipline of software engineering and is often categorised as *technology transfer*. One of the least well understood aspects of software development is in the move from requirements to design. We support the view that software designers fail to treat design as a process, and as a consequence become experts in representing the products using models/languages but fail to master the design process. Recent developments in academia have shown that design can be more effectively taught using problem based learning techniques. This appears to produce students who better understand design as a process; but how can we ensure

that this academic advancement will have a positive impact when these students move out into the real world?

Students act as one of the main conduits for technology transfer between academia and industry. Academics are responsible for ensuring that their students understand the most up-to-date theory and practice, and students are responsible for promoting these new techniques when they move out to an industrial setting.

SPLs are certainly a key future technology. It is critical that students are introduced to them when learning about design, architectures and model driven development. By integrating our research into formal SPLs into the teaching programme, we would also be exposing students to the power of formal methods.

4.9 Summary of Research Proposed: The Potential Impact — Résumé de la recherche proposée : l'impact potentiel

The next generation of electronic voting systems should be better engineered than the current generation. A first step towards this is a more thorough and precise analysis of the voting domain. Then, procurement offices can leverage the understanding in such a model in order to better specify their requirements. Consequently, manufacturers should be encouraged to develop a SPL for e-voting machines, in order to best manage the obvious commonalities and variations.

In this proposal, we have shown that a feature-oriented approach can be applied to the design of an e-voting SPL architecture. We believe that refinement has a key role to play in the development and application of such a SPL, particularly in the re-use of verified feature components. Integrating SPL techniques with formal methods is a promising approach: refinement for re-use of trustworthy components has already been addressed with respect to e-voting machine interfaces [CGM07b] and storage [CGM07a].

Current research — based on the notion of a feature interaction algebra[Gib98] — suggests that a *correct-by-construction* approach to guaranteeing the functionality of e-voting systems[CGM07b, CGM07a] merits further investigation.

Building an e-voting system has a high risk of failure due to unstable standards [GM08] and lack of understanding of the problem domain. Requirements creep has been a major problem

La prochaine génération de systèmes de vote électronique devra être mieux construite que la génération actuelle. Une analyse plus approfondie et plus précise du domaine du vote est une première étape vers ceci. Alors, les bureaux d'acquisition peuvent exercer une influence sur la compréhension d'un tel modèle dans le but de mieux spécifier leurs besoins. Par conséquent, les fabricants devraient être encouragés dans le developpement d'une SPL pour les machines de vote électronique, afin de mieux gérer les variations et points communs évidents.

Dans cette proposition, nous avons montré qu'une approche "feature-oriented" peut être appliquée au design d'une architecture de SPL pour le vote électronique. Nous croyons que le raffinement a un rôle-clé à jouer dans le développement et l'application d'une telle SPL, en particulier dans la réutilisation de composants *features* vérifiés. Intégrer des techniques de SPL avec des méthodes formelles est une approche prometteuse : le raffinement pour la réutilisation de composants fiables a déjà été traité en ce qui concerne les interfaces [CGM07b] et le stockage [CGM07a] d'une machine de vote électronique.

La recherche actuelle — fondée sur la notion d'un algèbre d'interactions de services [Gib98] — suggère qu'une approche "correct-by-construction" pour garantir la fonctionnalité des systèmes de vote électronique [CGM07b, CGM07a] mérite davantage d'investigation.

La construction d'un système de vote électronique comporte un risque élevé d'échec en raison de standards instables [GM08] et le manque de compréhension du domaine. "Requirements creep" constitue un problème majeur dans les systèmes de vote électronique. Ce qui con-

in e-voting systems. A good example is of the requirement for a voter verifiable audit trail (VVAT) for increased security [BPR+04]. Many current e-voting machines do not meet this requirement, and were not designed to do so. However, the election administrators and manufacturers seem to believe that this additional functionality can be somehow bolted on to already procured machines without risk. An e-voting SPL should be developed in order to manage the risk of evolving requirements: current research on maintaining SPLs [SE08] suggests that developing an e-voting SPL that can evolve as standards change is feasible using current techniques, but that it is a non-trivial problem. This is the main current and future research being proposed.

Although, we focus on e-voting, the fundamental research will have wider impact on software engineering in general; and formal methods and SPLs in particular.

The educational component of our proposal is a key part of our strategy for technology transfer. Without this component the potential impact of our proposed work could be compromised.

stitue un bon exemple, c'est celui de l'exigence pour un "voter verifiable audit trail" (VVAT) [BPR+04]. De nombreuses machines de vote électronique ne remplissent pas cette exigence et n'ont pas été conçues pour le faire. Néanmoins, les administrateurs d'élections et les fabricants semblent croire que cette fonctionnalité supplémentaire peut d'une manière ou d'une autre être boulonnée sans risque à des machines déjà procurées. Une SPL de vote électronique devrait être développée afin de gérer le risque d'exigences évoluents: la recherche actuelle sur le maintien des SPLs [SE08] suggèrent que développer une SPL de vote électronique pouvant évoluer en parallèle avec des standards qui changent est faisable en utilisant des techniques actuelles, mais que cela reste un problème pas banal. Ceci constitue la principale recherche actuelle et la future proposition de recherche.

Bien que nous nous concentrions sur le vote électronique, la recherche fondamentale aura un impact plus large sur le génie logiciel en général, et sur les méthodes formelles et les SPLs en particulier.

Le composant pédagogique de notre proposition est un élément clé de notre stratégie pour le transfert de technologie. Sans ce composant, l'impact potentiel de notre proposition de travail pourrait être compromis.

4.A-Research Proposal For A Formal SPL for E-voting: Bibliography

[ABHV06] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, and Laurent Voisin. An open extensible tool environment for Event-B. In Zhiming Liu and Jifeng He, editors, *Formal Methods and Software Engineering, 8th International Conference on Formal Engineering Methods, ICFEM 2006*, volume 4260 of *Lecture Notes in Computer Science*, pages 588–605, Macao, China, 2006. Springer.

[Abr07] Jean-Raymond Abrial. A system development process with Event-B and the Rodin platform. In Michael Butler, Michael G. Hinchey, and María M. Larrondo-Petrie, editors, *Formal Methods and Software Engineering, 9th International Conference on Formal Engineering Methods, ICFEM 2007*, volume 4789 of *Lecture Notes in Computer Science*, pages 1–3, Boca Raton, FL, USA, 2007. Springer.

[ACC+08] Adam Aviv, Pavol Cerný, Sandy Clark, Eric Cronin, Gaurav Shah, Micah Sherr, and Matt Blaze. Security evaluation of ES&S voting machines and election management system. In *EVT'08: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2008*, Berkeley, CA, USA, July 2008. USENIX Association.

[AFT07] R. Anane, R. Freeland, and G. Theodoropoulos. E-voting requirements and implementation. In *The 9th IEEE International Conference on E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007. CEC/EEE*, pages 382–392, Tokyo, Japan, July 2007.

[AH07] Jean-Raymond Abrial and Stefan Hallerstede. Refinement, decomposition, and instantiation of discrete models: Application to event-b. *Fundam. Inf.*, 77(1-2):1–28, 2007.

[AM05] Chris Armen and Ralph Morelli. Teaching about the risks of electronic voting technology. *SIGCSE Bull.*, 37(3):227–231, 2005.

[AR06] Ben Adida and Ronald L. Rivest. Scratch & vote: self-contained paper-based cryptographic voting. In Ari Juels and Marianne Winslett, editors, *Workshop on Privacy in the Electronic Society WPES*, pages 29–40, Alexandria, VA, USA, 2006. ACM.

[ASH+08] Nirwan Ansari, Pitipatana Sakarindr, Ehsan Haghani, Chao Zhang, Aridaman K. Jain, and Yun Q. Shi. Evaluating electronic voting systems equipped with voter-verified paper records. *IEEE Security and Privacy*, 6(3):30–39, 2008.

[AW97] Mark A. Ardis and David M. Weiss. Defining families: the commonality analysis (tutorial). In *ICSE '97: Proceedings of the 19th international conference on Software engineering*, pages 649–650, New York, NY, USA, 1997. ACM.

[BB06] Nadja Braun and Daniel Brändli. Swiss e-voting pilot projects: Evaluation, situation analysis and how to proceed. In Krimmer [Kri06], pages 27–36.

[BBC+08] Davide Balzarotti, Greg Banks, Marco Cova, Viktoria Felmetsger, Richard A. Kemmerer, William Robertson, Fredrik Valeur, and Giovanni Vigna. Are your votes *really* counted?: Testing the security of real-world electronic voting systems. In Barbara G. Ryder and Andreas Zeller, editors, *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis ISSTA*, pages 237–248, Seattle, WA, USA, 2008. ACM.

[BEF+56] B. S. Bloom, M. D. Engelhart, E. J. Furst, W. H. Hill, and D. R. Krathwohl. *Taxonomy of educational objectives Handbook 1: cognitive domain*. Longman Group Ltd., London, 1956.

[BEH+08] Kevin Butler, William Enck, Harri Hursti, Stephen McLaughlin, Patrick Traynor, and Patrick McDaniel. Systemic Issues in the Hart InterCivic and Premier Voting Systems: Reflections on Project EVEREST. In *EVT'08: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2008 on Electronic Voting Technology Workshop*, Berkeley, CA, USA, July 2008. USENIX Association.

[Ben07] Josh Benaloh. Ballot casting assurance via voter-initiated poll station auditing. In *EVT'07: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2007 on Electronic Voting Technology Workshop*, Berkeley, CA, USA, August 2007. USENIX Association.

[BF07] Matt Bishop and Deborah A. Frincke. Achieving learning objectives through e-voting case studies. *IEEE Security and Privacy*, 5(1):53–56, 2007.

[BFK+99] Joachim Bayer, Oliver Flege, Peter Knauber, Roland Laqua, Dirk Muthig, Klaus Schmid, Tanya Widen, and Jean-Marc DeBaud. Pulse: A methodology to develop software product lines. In *SSR*, pages 122–131, 1999.

[BFMV07] Volha Bryl, Roberta Ferrario, Andrea Mattioli, and Adolfo Villafiorita. Evaluating Procedural Alternatives in an e-Voting Domain: Lessons Learned. Technical Report DIT-07-005, University of Trento, DIT, Italy, 2007.

[BGE07] Michael D. Byrne, Kristen K. Greene, and Sarah P. Everett. Usability of voting systems: baseline data for paper, punch cards, and lever machines. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 171–180, New York, USA, 2007. ACM.

[BL05] Michael J. Butler and Michael Leuschel. Combining csp and b for specification and property verification. In John Fitzgerald, Ian J. Hayes, and Andrzej Tarlecki, editors, *FM*, volume 3582 of *Lecture Notes in Computer Science*, pages 221–236. Springer, 2005.

[BLRS06] J W. Bryans, B Littlewood, P Y. A. Ryan, and L Strigini. E-voting: Dependability requirements and design for dependability. In *ARES '06: Proceedings of the First International Conference on Availability, Reliability and Security*, pages 988–995, Washington, DC, USA, 2006. IEEE Computer Society.

[BLS+03] Benjamin B. Bederson, Bongshin Lee, Robert M. Sherman, Paul S. Herrnson, and Richard G. Niemi. Electronic voting system usability issues. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 145–152, New York, USA, 2003. ACM.

[Bos99a] Jan Bosch. Evolution and composition of reusable assets in product-line architectures: A case study. In *WICSA1: Proceedings of the TC2 First Working IFIP Conference on Software Architecture (WICSA1)*, pages 321–340, Deventer, The Netherlands, The Netherlands, 1999. Kluwer, B.V.

[Bos99b] Jan Bosch. Product-line architectures in industry: a case study. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 544–554, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[BPR+04] Jonathan Bannet, David W. Price, Algis Rudys, Justin Singer, and Dan S. Wallach. Hack-a-vote: Security issues with electronic voting systems. *IEEE Security & Privacy*, 2(1):32–37, 2004.

[BR96] Joan Borrell and Josep Rifà. An implementable secure voting scheme. *Computers & Security*, 15(4):327–338, 1996.

[Bra78] C. Brainerd. *Piaget's Theory of Intelligence*. Prentice Hall, Englewood Cliffs, NJ, 1978.

[Bre06] Peter Brent. The Australian ballot: Not the secret ballot. *Australian Journal of Political Science*, 41(1):39–50, March 2006.

[Bru66] J. S. Bruner. *Toward a theory of instruction*. Belknap Press of Harvard University, Cambridge, Mass,, 1966.

[BT80] H.S. Barrows and R.M. Tamblyn. *Problem-Based Learning: An Approach to Medical Education*. Springer Publishing Company, New York, 1980.

[BT94] Josh Cohen Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *Twenty-Sixth Annual ACM Symposium on Theory of Computing STOC*, pages 544–553, Montréal, Québec, Canada, May 1994.

[BY86] Josh Cohen Benaloh and Moti Yung. Distributing the power of a government to enhance the privacy of voters (extended abstract). In *Fifth Annual ACM Symposium on Princiles of Distributed ComputingPODC*, pages 52–62, Calgary, Alberta, Canada, August 1986.

[CC97] Lorrie Faith Cranor and Ron K. Cytron. Sensus: A security-conscious electronic polling system for the internet. In *30th Hawaii International Conference on System Sciences (HICSS) Volume 3*, pages 561–570. IEEE Computer Society, 1997.

[CC07] O. Cetinkaya and D. Cetinkaya. Verification and validation issues in electronic voting. *The Electronic Journal of e-Government*, 5(2):117–126, 2007.

[CCM07] Michael E. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: A secure remote voting system. In Chaum et al. [CKRR08].

[CEB+05] Deirdre Carew, Chris Exton, Jim Buckley, Margaret McGaley, and J.Paul Gibson. Preliminary study to empirically investigate the comprehensibility of requirements specifications. In *Psychology of Programming Interest Group 17th annual workshop (PPIG)*, pages 182–202, University of Sussex, Brighton, UK, 2005.

[CEC+08] D. Chaum, A. Essex, R. Carback, J. Clark, S. Popoveniuc, A. Sherman, and P. Vora. Scantegrity: End-to-end voter-verifiable optical-scan voting. *Security & Privacy*, 6(3):40–46, May/June 2008.

[CF85] Josh D. Cohen and Michael J. Fischer. A robust and verifiable cryptographically secure election scheme (extended abstract). In *26th Annual Symposium on Foundations of Computer Science(FOCS)*, pages 372–382, Portland, Oregon, USA, October 1985. IEEE.

[CFM+08] Stefano Campanelli, Alessandro Falleni, Fabio Martinelli, Marinella Petrocchi, and Anna Vaccarelli. Mobile implementation and formal verification of an e-voting system. In Abdelhamid Mellouk, Jun Bi, Guadalupe Ortiz, Dickson K. W. Chiu, and Manuela Popescu, editors, *Third International Conference on Internet and Web Applications and Services (ICIW)*, pages 476–481, Athens, Greece, June 2008. IEEE Computer Society.

[CGM07a] Dominique Cansell, J. Paul Gibson, and Dominique Méry. Formal verification of tamper-evident storage for e-voting. In *SEFM*, pages 329–338. IEEE Computer Society, 2007.

[CGM07b] Dominique Cansell, J. Paul Gibson, and Dominique Méry. Refinement: A constructive approach to formal software design for a secure e-voting interface. *Electr. Notes Theor. Comput. Sci.*, 183:39–55, 2007.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT*, pages 103–118, Konstanz, Germany, May 1997.

[Cha81] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[Cha04] David Chaum. Secret-Ballot Receipts: True Voter-Verifiable Elections. In *Security & Privacy (Vol. 2, No. 1)*, pages 38–47. IEEE, January/February 2004.

[CHS+10] Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens, Axel Legay, and Jean-François Raskin. Model checking lots of systems: Efficient verification of temporal properties in software product lines (to appear). In *32nd International Conference on Software Engineering, ICSE 2010, May 2-8, 2010, Cape Town, South Africa, Proceedings*. IEEE, 2010.

[CJC04] Yu-Yi Chen, Jinn-Ke Jan, and Chin-Ling Chen. The design of a secure anonymous internet voting system. *Computers & Security*, 23(4):330–337, 2004.

[CKRR08] David Chaum, Miroslaw Kutylowski, Ronald L. Rivest, and Peter Y. A. Ryan, editors. *Frontiers of Electronic Voting, 29.07. - 03.08.2007*, volume 07311 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2008.

[CN02] P. Clements and L. Northrop. *Software product lines*. Addison-Wesley Boston, 2002.

[Cra96] Lorrie Faith Cranor. Electronic voting: computerized polls may save money, protect privacy. *Crossroads*, 2(4):12–16, 1996.

[Cra01] Lorrie Faith Cranor. Voting after Florida: no easy answers. *Ubiquity*, 1(47):1, 2001.

[CRS05] David Chaum, Peter Y. A. Ryan, and Steve A. Schneider. A practical voter-verifiable election scheme. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *10th European Symposium On Research In Computer Security(ESORICS)*, volume 3679 of *Lecture Notes in Computer Science*, pages 118–139, Milan, Italy, September 2005. Springer.

[Cur03] W. S. Curran. Teaching software engineering in the computer science curriculum. *SIGCSE Bull.*, 35(4):72–75, 2003.

[DD07] Scott G Decker and Jim Dager. Software product lines beyond software development. In *SPLC '07: Proceedings of the 11th International Software Product Line Conference*, pages 275–280, Washington, DC, USA, 2007. IEEE Computer Society.

[Dhu06] Deepak Dhungana. Integrated variability modeling of features and architecture in software product line engineering. In *ASE*, pages 327–330. IEEE Computer Society, 2006.

[Dij72] Edsger W. Dijkstra. *Structured programming*, chapter Notes on structured programming, pages 1–82. Academic Press Ltd., London, UK, 1972.

[DJOS07] Jim Davies, Tomasz Janowski, Adegboyega Ojo, and Aadya Shukla. Technological foundations of electronic governance. In *ICEGOV '07: Proceedings of the 1st international conference on Theory and practice of electronic governance*, pages 5–11, New York, NY, USA, 2007. ACM.

[DKK+08] Seda Davtyan, Sotiris Kentros, Aggelos Kiayias, Laurent Michel, Nicolas Nicolaou, Alexander Russell, Andrew See, Narasimha Shashidhar, and Alexander A. Shvartsman. Pre-election testing and post-election audit of optical scan voting terminal memory cards. In *EVT'08: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop*, Berkeley, CA, USA, July 2008. USENIX Association.

[DKR06] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *Proceedings of the 19th IEEE workshop on Computer Security Foundations (CSFW)*, pages 28–42. IEEE Computer Society, 2006.

[DLM82] Richard A. DeMillo, Nancy A. Lynch, and Michael Merritt. Cryptographic protocols. In *14th Annual ACM Symposium on Theory of Computing (STOC)*, pages 383–400, San Francisco, California, USA, May 1982. ACM.

[dMPQ07] Olivier de Marneffe, Olivier Pereira, and Jean-Jacques Quisquater. Simulation-based analysis of e2e voting systems. In Chaum et al. [CKRR08].

[DS99] Jean-Marc DeBaud and Klaus Schmid. A systematic approach to derive the scope of software product lines. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 34–43, New York, NY, USA, 1999. ACM.

[EB08] Andrew Edmunds and Michael Butler. Linking event-b and concurrent object-oriented programs. *Electr. Notes Theor. Comput. Sci.*, 214:159–182, 2008.

[EBB06] Magnus Eriksson, Jürgen Börstler, and Kjell Borg. Software product line modeling made practical. *Commun. ACM*, 49(12):49–54, 2006.

[Eve07] Sarah P. Everett. *The Usability of Electronic Voting Machines and How Votes Can Be Changed Without Detection*. PhD thesis, Rice University, Houston, TX, USA, 2007.

[FHF07] Ariel J. Feldman, J. Alex Halderman, and Edward W. Felten. Security analysis of the Diebold AccuVote-TS voting machine. In *EVT'07: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2007*, Berkeley, CA, USA, August 2007. USENIX Association.

[FOO93] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In *ASIACRYPT '92: Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques*, volume 718 of *Lecture Notes in Computer Science*, pages 244–251, London, UK, 1993. Springer-Verlag.

[Gar83] H. Gardner. *Frames of mind: the theory of multiple intelligence*. Basic Books, New York, 1983.

[GGR07] Ryan Gardner, Sujata Garera, and Aviel D. Rubin. On the difficulty of validating voting machine software with software. In *EVT'07: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2007*, Berkeley, CA, USA, August 2007. USENIX Association.

[GH07] Rop Gonggrijp and Willem-Jan Hengeveld. Studying the Nedap/Groenendaal ES3B voting computer: A computer security perspective. In *EVT'07: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2007*, Berkeley, CA, USA, August 2007. USENIX Association.

[Gib98] J. Paul Gibson. Towards a feature interaction algebra. In Kristofer Kimbler and Wiet Bouma, editors, *Feature Interactions in Telecommunications and Software Systems V (FIW 1998)*, pages 217–231, Malmö, Sweden, 1998. IOS Press.

[Gib00] J. Paul Gibson. Formal requirements engineering: Learning from the students. In Doug Grant, editor, *12th Australian Software Engineering Conference (ASWEC 2000)*, pages 171–180. IEEE Computer Society, 2000.

[Gib07] J. Paul Gibson. E-voting and the need for rigourous software engineering - the past, present and future. In Jacques Julliand and Olga Kouchnarenko, editors, *B 2007*, volume 4355 of *Lecture Notes in Computer Science*, page 1. Springer, 2007.

[Gib08a] J. Paul Gibson. Formal methods - never too young to start. In *Formal Methods in Computer Science Education (FORMED)*, pages 149–159, March 2008.

[Gib08b] J. Paul Gibson. Weaving a formal methods education with problem-based learning. In T. Margaria and B. Steffen, editors, *3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*, volume 17 of *Communications in Computer and Information Science (CCIS)*, pages 460–472, Porto Sani, Greece, October 2008. Springer-Verlag, Berlin Heidelberg.

[GJ09] J. Paul Gibson and Doug Jones, editors. *First International Workshop on Requirements Engineering for e-Voting Systems (RE-VOTE09)*, Atlanta, GA, USA, August 2009. IEEE.

[GKK+06] Marcin Gogolewski, Marek Klonowski, Przemyslaw Kubiak, Miroslaw Kutylowski, Anna Lauks, and Filip Zagórski. Kleptographic attacks on e-voting schemes. In Günter Müller, editor, *International Conference on Emerging Trends in Information and Communication Security (ETRICS)*, volume 3995 of *Lecture Notes in Computer Science*, pages 494–508. Springer, 2006.

[GLR08a] J. Paul Gibson, Eric Lallet, and Jean-Luc Raffy. Analysis of a distributed e-voting system architecture against quality of service requirements. In Herwig Mannaert, Tadashi Ohta, Cosmin Dini, and Robert Pellerin, editors, *The Third International Conference on Software Engineering Advances (ICSEA 2008)*, pages 58–64, Sliema, Malta, October 2008. IEEE Computer Society.

[GLR08b] J. Paul Gibson, Eric Lallet, and Jean-Luc Raffy. How do I know if my design is correct? In Zoltan Istenes, editor, *Formal Methods in Computer Science Education (FORMED 2008)*, pages 61–70, Budapest, Hungary, March 2008. Accepted for publication in ENTCS.

[GLR10] J. Paul Gibson, Eric Lallet, and Jean-Luc Raffy. Engineering a distributed e-voting system architecture: Meeting critical requirements. In Holger Giese, editor, *Architecting Critical Systems, First International Symposium, ISARCS 2010, Prague, Czech Republic, June 23-25, 2010, Proceedings*, volume 6150 of *Lecture Notes in Computer Science*, pages 89–108. Springer, 2010.

[GM98] J. Paul Gibson and Dominique Mery. Teaching formal methods: Lessons to learn. In Sharon Flynn and Andrew Butterfield, editors, *2nd Irish Workshop on Formal Methods (IWFM 1998)*, Electronic Workshops in Computing, Cork, Ireland, July 1998. BCS.

[GM08] J. Paul Gibson and Margaret McGaley. Verification and maintenance of e-voting systems and standards. In Dan Remenyi, editor, *8th European Conference on e-Government*, pages 283–289. Academic Publishing International, July 2008. ISBN 978-1-906638-09-2.

[GO05] J. Paul Gibson and Jackie O'Kelly. Software engineering as a model of understanding for learning and problem solving. In *ICER '05: Proceedings of the 2005 international workshop on Computing education research*, pages 87–97, New York, NY, USA, 2005. ACM.

[Got06] Don Gotterbarn. E-voting: a failure of professionalism? In *ITiCSE-WGR '06: Working group reports on ITiCSE on Innovation and technology in computer science education*, pages 7–8, New York, USA, 2006. ACM.

[Gri02] Dimitris Gritzalis. Principles and requirements for a secure e-voting system. *Computers & Security*, 21(6):539–556, 2002.

[Gri03] Dimitris Gritzalis, editor. *Secure Electronic Voting*, volume 7 of *Advances in Information Security*. Kluwer Academic, 2003.

[Gro04a] Jens Groth. Evaluating security of voting schemes in the universal composability framework. In Markus Jakobsson, Moti Yung, and Jianying Zhou, editors, *Applied Cryptography and Network Security*, volume 3089 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 2004.

[Gro04b] Jeff Grove. ACM statement on voting systems. *Communications of the ACM*, 47(10):69–70, 2004.

[Gui67] J. P. Guilford. *The Nature of Human Intelligence*. McGraw-Hill, New York, 1967.

[Hab08] Henri Habrias. Teaching specifications, hands on. In *Formal Methods in Computer Science Education (FORMED)*, pages 5–15, March 2008.

[Hal08] Joseph Lorenzo Hall. Improving the security, transparency, and efficiency of California's 1% manual tally procedures. In *EVT'08: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop*, Berkeley, CA, USA, July 2008. USENIX Association.

[HB56] E. R. Hilgard and G. H. Bower. *Theories of Learning*. Prentice Hall, Englewood Cliffs, NJ, 1956.

[HBL+05] Paul S. Herrnson, Benjamin B. Bederson, Bongshin Lee, Peter L. Francia, Robert M. Sherman, Frederick G. Conrad, Michael Traugott, and Richard G. Niemi. Early appraisals of electronic voting. *Social Science Computer Review*, 23(3):274–292, 2005.

[Hea07] James Heather. Implementing STV securely in Prêt à Voter. In *20th IEEE Computer Security Foundations Symposium(CSF)*, pages 157–169, Venice, Italy, July 2007. IEEE Computer Society.

[HGM00] Geoff Hamilton, J. Paul Gibson, and Dominique Méry. Composing fair objects. In Fouchal and Lee, editors, *International Conference on Software Engineering Applied to Networking and Parallel/Distributed Computing (SNPD '00)*, pages 225–233, Reims, France, May 2000.

[HV07] Benjamin Hosp and Poorvi L. Vora. An information-theoretic model of voting systems. In Chaum et al. [CKRR08].

[Hwa96] Jing-Jang Hwang. A conventional approach to secret balloting in computer networks. *Computers and Security*, 15(3):249–263, 1996.

[HWW87] I. D. Hill, B. A. Wichmann, and D. R. Woodall. Algorithm 123 — Single transferable vote by Meek's method. *Computer Journal*, 30:277–281, 1987.

[Ive91] Kenneth R. Iversen. A cryptographic scheme for computerized elections. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91, 11th Annual International Cryptology Conference*, volume 576 of *Lecture Notes in Computer Science*, pages 405–419, Santa Barbara, California, USA, 1991. Springer.

[JCJ05] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In Vijay Atluri, Sabrina De Capitani di Vimercati, and Roger Dingledine, editors, *Workshop on Privacy in the Electronic Society (WPES)*, pages 61–70, Alexandria, VA, USA, 2005. ACM.

[Jef07] David Jefferson. What happened in Sarasota county? *The Bridge: Linking Engineering and Society*, 37(2):17–23, 2007.

[JRSW04a] David Jefferson, Aviel D. Rubin, Barbara Simons, and David Wagner. Analyzing internet voting security. *Commun. ACM*, 47(10):59–64, 2004.

[JRSW04b] David Jefferson, Aviel D. Rubin, Barbara Simons, and David Wagner. Analyzing internet voting security. *Communications of the ACM*, 47(10):59–64, 2004.

[JZF03] Rui Joaquim, Andre Zuquete, and Paulo Ferreira. REVS - A Robust Electronic Voting System. In *Proceedings of the IADIS International Conference on e-Society*, pages 95–103, Lisbon, Portugal, June 2003.

[KCH+90] K.C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU-SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.

[KCT07] Joseph R. Kiniry, Dermot Cochran, and Patrick E. Tierney. Verification-centric realization of electronic vote counting. In *EVT'07: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2007 on Electronic Voting Technology Workshop*, Berkeley, CA, USA, august 2007. USENIX Association.

[KE03] Chethana Kuloor and Armin Eberlein. Aspect-oriented requirements engineering for software product lines. In *ECBS*, pages 98–107. IEEE Computer Society, 2003.

[Kin07] Joseph Kiniry. Formally counting electronic votes (but still only trusting paper). In *ICECCS*, pages 261–269. IEEE Computer Society, 2007.

[KLD02] Kyo C. Kang, Jaejoon Lee, and Patrick Donohoe. Feature-oriented product line engineering. *IEEE Software*, 19(4):58–65, 2002.

[KMC+06] Joseph R. Kiniry, Alan E. Morkan, Dermot Cochran, Martijn Oostdijk, and Engelbert Hubbers. Formal techniques in a remote voting system. *SIGSOFT Softw. Eng. Notes*, 31(6):1–2, 2006.

[KMR+07a] Aggelos Kiayias, Laurent Michel, Alexander Russell, Narasimha Sashidar, Andrew See, and Alexander A. Shvartsman. An authentication and ballot layout attack against an optical scan voting terminal. In *EVT'07: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2007 on Electronic Voting Technology Workshop*, Berkeley, CA, USA, august 2007. USENIX Association.

[KMR+07b] Aggelos Kiayias, Laurent Michel, Alexander Russell, Narasimha Shashidhar, Andrew See, Alexander A. Shvartsman, and Seda Davtyan. Tampering with special purpose trusted computing devices: A case study in optical scan e-voting. In *ACSAC*, pages 30–39. IEEE Computer Society, 2007.

[KR05] Steve Kremer and Mark Ryan. Analysis of an electronic voting protocol in the applied pi-calculus. In *In European Symposium on Programming, number 3444 in Lecture Notes in Computer Science*, pages 186–200. Springer, 2005.

[Kri06] Robert Krimmer, editor. *Electronic Voting 2006: 2nd International Workshop, Co-organized by Council of Europe, ESF TED, IFIP WG 8.6 and E-Voting.CC, August, 2nd - 4th, 2006 in Castle Hofen, Bregenz, Austria*, volume 86 of *LNI*. GI, 2006.

[KS04] Paul Kocher and Bruce Schneier. Insider risks in elections. *Commun. ACM*, 47(7):104, 2004.

[KSRW04] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan S. Wallach. Analysis of an electronic voting system. In *IEEE Symposium on Security and Privacy (S&P04)*, pages 27–40. IEEE, 2004.

[KSW05] Chris Karlof, Naveen Sastry, and David Wagner. Cryptographic voting protocols: a systems perspective. In *SSYM'05: Proceedings of the 14th conference on USENIX Security Symposium*, pages 3–3, Berkeley, CA, USA, 2005. USENIX Association.

[KTV07] Robert Krimmer, Stefan Triessnig, and Melanie Volkamer. The development of remote e-voting around the world: A review of roads and directions. In Ammar Alkassar and Melanie Volkamer, editors, *VOTE-ID*, volume 4896 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2007.

[KY04] Aggelos Kiayias and Moti Yung. The vector-ballot e-voting approach. In Ari Juels, editor, *Financial Cryptography*, volume 3110 of *Lecture Notes in Computer Science*, pages 72–89. Springer, 2004.

[KZ08] Joseph R. Kiniry and Daniel M. Zimmerman. Secret ninja formal methods. In Jorge Cuéllar, T. S. E. Maibaum, and Kaisa Sere, editors, *FM*, volume 5014 of *Lecture Notes in Computer Science*, pages 214–228. Springer, 2008.

[Lar99] Kai R. T. Larsen. Voting technology implementation. *Commun. ACM*, 42(12):55–57, 1999.

[Lau04] Thomas W. Lauer. The risk of e-voting. *Electronic Journal of e-Government*, 2:177–186, 2004.

[Liu07] Jing Liu. Handling safety-related feature interaction in safety-critical product lines. In *ICSE Companion*, pages 85–86. IEEE Computer Society, 2007.

[Lun07] David Lundin. Component based electronic voting systems. In Chaum et al. [CKRR08].

[Maa04] Epp Maaten. Towards remote e-voting: Estonian case. In Alexander Prosser and Robert Krimmer, editors, *Electronic Voting in Europe*, volume 47 of *LNI*, pages 83–100. GI, 2004.

[MC04] Rebecca T. Mercuri and L. Jean Camp. The code of elections. *Communications of the ACM*, 47(10):52–57, 2004.

[McI68] D. McIlroy. Mass-produced software components. In *Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany*, pages 88–98, 1968.

[Mer92] Rebecca Mercuri. Voting-machine risks. *Commun. ACM*, 35(11):138, 1992.

[Mer93] Rebecca Mercuri. Corrupted polling. *Commun. ACM*, 36(11):122, 1993.

[Mer02a] Rebecca Mercuri. Florida 2002: sluggish systems, vanishing votes. *Commun. ACM*, 45(11):136, 2002.

[Mer02b] Rebecca Mercuri. Government: a better ballot box? *IEEE Spectr.*, 39(10):46–50, 2002.

[MG03] Margaret McGaley and J. Paul Gibson. E-voting: a safety critical system. Report NUIM-CS-TR-2003-2, Department of Computer Science, National University of Ireland, Maynooth, 2003.

[MG06] Margaret McGaley and J. Paul Gibson. A critical analysis of the council of europe recommendations on e-voting. In *EVT'06: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, pages 9–22, Berkeley, CA, USA, 2006. USENIX Association.

[MKSW06] D. Molnar, T. Kohno, N. Sastry, and D. Wagner. Tamper-evident, history-independent, subliminal-free data structures on prom storage — or — how to store ballots on a voting machine (extended abstract). *IEEE Symposium on Security and Privacy,*, 2006.

[MLF06] Rebecca T. Mercuri, Vincent J. Lipsio, and Beth Feehan. COTS and other electronic voting backdoors. *Commun. ACM*, 49(11):112, 2006.

[MM04] Margaret McGaley and Joe McCarthy. Transparency and e-Voting: Democratic vs. Commercial Interests. In *Electronic Voting in Europe - Technology, Law, Politics and Society*, pages 153 – 163. European Science Foundation, July 2004.

[MM06] Ülle Madise and Tarvi Martens. E-voting in estonia 2005. the first practice of country-wide binding internet voting in the world. In Krimmer [Kri06], pages 15–26.

[MW93] P. Mukherjee and B. Wichmann. STV: A Case Study in VDM. Technical report, National Physical Laboratory, Teddington, UK, 1993.

[MW95] Paul Mukherjee and Brian A. Wichmann. Formal specification of the stv algorithm. In Jonathan P. Bowen and Michael G. Hinchey, editors, *Applications of Formal Methods*, pages 73–96, Upper Saddle River, NJ, USA, 1995. Prentice Hall.

[MWC09] Marcilio Mendonca, Andrzej Wasowski, and Krzysztof Czarnecki. SAT-Based analysis of feature models is easy. In *Proceedings of the 13th International Software Product Lines Conference (SPLC'09), San Francisco, CA, USA*, pages 231–240, 2009.

[Nef01] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *ACM Conference on Computer and Communications Security*, pages 116–125, 2001.

[Neu90] Peter G. Neumann. Inside risks: risks in computerized elections. *Commun. ACM*, 33(11):170, 1990.

[Neu04] Peter G. Neumann. The problems and potentials of voting systems. *Commun. ACM*, 47(10), 2004.

[Neu05] Peter G. Neumann. Responsibilities of technologists. *Commun. ACM*, 48(2):128, 2005.

[NR95] Valtteri Niemi and Ari Renvall. How to prevent buying of votes in computer elections. In *ASIACRYPT '94: Proceedings of the 4th International Conference on the Theory and Applications of Cryptology*, pages 164–170, London, UK, 1995. Springer-Verlag.

[NR99] Valtteri Niemi and Ari Renvall. Efficient voting with no selling of votes. *Theor. Comput. Sci.*, 226(1-2):105–116, 1999.

[NS94] Hannu Nurmi and Arto Salomaa. Conducting secret ballot elections in computer networks: Problems and solutions. *Annals of Operations Research*, 51(4):185–194, 1994.

[OG05] J. O'Kelly and J. Paul Gibson. PBL: Year one analysis — interpretation and validation. In *PBL In Context — Bridging Work and Education*, 2005.

[PAB+04] Kun Peng, Riza Aditya, Colin Boyd, Ed Dawson, and Byoungcheon Lee. Multiplicative homomorphic e-voting. In Anne Canteaut and Kapalee Viswanathan, editors, *INDOCRYPT*, volume 3348 of *Lecture Notes in Computer Science*, pages 61–72. Springer, 2004.

[Par76] David Lorge Parnas. On the design and development of program families. *IEEE Trans. Software Eng.*, 2(1):1–9, 1976.

[PFF+08] M. Poppleton, B. Fischer, C. Franklin, A. Gondal, C. Snook, and J. Sorge. Towards reuse with "feature-oriented event-b". In *McGPLE: Workshop on Modularization, Composition, and Generative Techniques for Product Line Engineering*, pages 1–6. Department of Informatics and Mathematics University of Passau, Germany, October 2008.

[PGL00] Gilda Pour, Martin L. Griss, and Michael J. Lutz. The push to make software engineering respectable. *IEEE Computer*, 33(5):35–43, 2000.

[Pop97] Michael Poppleton. The single transferable voting system: Functional decomposition in formal specification. In Henry McGloughlin and Gerard O'Regan, editors, *IWFM*, Workshops in Computing. BCS, 1997.

[Pop07] Michael Poppleton. Towards feature-oriented specification and development with event-b. In Peter Sawyer, Barbara Paech, and Patrick Heymans, editors, *REFSQ*, volume 4542 of *Lecture Notes in Computer Science*, pages 367–381. Springer, 2007.

[Pop08] Michael Poppleton. The composition of event-b models. In *ABZ '08: Proceedings of the 1st international conference on Abstract State Machines, B and Z*, pages 209–222, Berlin, Heidelberg, 2008. Springer-Verlag.

[PRH+07] Elliot Proebstel, Sean Riddle, Francis Hsu, Justin Cummins, Freddie Oakley, Tom Stanionis, and Matt Bishop. An analysis of the hart Intercivic DAU eSlate. In *EVT'07: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2007 on Electronic Voting Technology Workshop*, Berkeley, CA, USA, august 2007. USENIX Association.

[PS80] S. Papert and J. Sculley. *Mindstorms: children,computers, and powerful ideas*. Basic Books, New York, 1980.

[PS06] David Lorge Parnas and Michael Soltys. Basic science for software developers. In *Workshop on Formal Methods in the Teaching Lab (FM-Ed 2006*, pages 9–14, August 2006.

[QT07] Ghassan Z. Qadah and Rani Taha. Electronic voting systems: Requirements, design, and implementation. *Comput. Stand. Interfaces*, 29(3):376–386, 2007.

[RFAB06] José Rodrigues-Filho, Cynthia Alexander, and Luciano Batista. E-voting in brazil - the risks to democracy. In Krimmer [Kri06], pages 85–94.

[RH07] Thomas P. Ryan and Candice Hoke. GEMS tabulation database design issues in relation to voting systems certification standards. In *EVT'07: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2007 on Electronic Voting Technology Workshop*, Berkeley, CA, USA, august 2007. USENIX Association.

[Rot98] Susan King Roth. Disenfranchised by design: voting systems and the election process. *Information Design Journal*, 9(1):1–8, 1998.

[RP05] Peter Y. A. Ryan and Thea Peacock. Prêt à voter: a systems perspective. Technical Report CS-TR-929, University of Newcastle, 2005.

[RR06] Brian Randell and Peter Y. A. Ryan. Voting technologies and trust. *IEEE Security and Privacy*, 4(5):50–56, 2006.

[RS07] Ronald L. Rivest and Warren D. Smith. Three voting protocols: ThreeBallot, VAV, and Twin. In *EVT'07: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2007 on Electronic Voting Technology Workshop*, Berkeley, CA, USA, august 2007. USENIX Association.

[RTS07] Ben Riva and Amnon Ta-Shma. Bare-handed electronic voting with pre-processing. In *EVT'07: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2007 on Electronic Voting Technology Workshop*, Berkeley, CA, USA, august 2007. USENIX Association.

[Rub02] Aviel D. Rubin. Security considerations for remote electronic voting. *Commun. ACM*, 45(12):39–44, 2002.

[SB08] Colin F. Snook and Michael J. Butler. Uml-b: A plug-in for the event-b tool set. In Egon Börger, Michael J. Butler, Jonathan P. Bowen, and Paul Boca, editors, *ABZ*, volume 5238 of *Lecture Notes in Computer Science*, page 344. Springer, 2008.

[SBS09] Mar Yah Said, Michael J. Butler, and Colin F. Snook. Language and tool support for class and state machine refinement in uml-b. In Ana Cavalcanti and Dennis Dams, editors, *FM*, volume 5850 of *Lecture Notes in Computer Science*, pages 579–595. Springer, 2009.

[Sch85] A. H. Schoenfeld. *Mathematical Problem Solving*. Academic Press, Orlando, Fla, 1985.

[Sch99] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 148–164. Springer, 1999.

[Sch00] Berry Schoenmakers. Compensating for a lack of transparency. In *CFP '00: Proceedings of the tenth conference on Computers, freedom and privacy*, pages 231–233, New York, NY, USA, 2000. ACM.

[SD04] Tim Storer and Ishbel Duncan. Practical remote electronic elections for the uk. In *PST*, pages 41–45, 2004.

[SD05] Tim Storer and Ishbel Duncan. Electronic voting in the uk: Current trends in deployment, requirements and technologies. In *PST*, 2005.

[SE08] Klaus Schmid and Holger Eichelberger. A requirements-based taxonomy of software product line evolution. *Electronic Communications of the EASST*, 8:2–13, 2008. Software Evolution 2007.

[SG04] T. Selker and J. Goler. The save system — secure architecture for voting electronically. *BT Technology Journal*, 22(4):89–95, 2004.

[Sha00] David C. Sharp. Component-based product line development of avionics software. In Patrick Donohoe, editor, *SPLC*, pages 353–370. Kluwer, 2000.

[SK95] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme - a practical solution to the implementation of a voting booth. In *EUROCRYPT*, pages 393–403, 1995.

[SKW06] Naveen Sastry, Tadayoshi Kohno, and David Wagner. Designing voting machines for verification. In *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium*, pages 22–22, Berkeley, CA, USA, 2006. USENIX Association.

[SL03] Jörgen Svensson and Ronald Leenes. E-voting in europe: Divergent democratic practice. *Information Polity*, 8(1):3–15, 2003.

[SMCO08] Borislava I. Simidchieva, Matthew S. Marzilli, Lori A. Clarke, and Leon J. Osterweil. Specifying and verifying requirements for election processes. In *dg.o'08: Proceedings of the 2008 international conference on Digital government research*, pages 63–72. Digital Government Society of North America, 2008.

[SP06] Krishna Sampigethaya and Radha Poovendran. A framework and taxonomy for comparison of electronic voting schemes. *Computers & Security*, 25(2):137–153, 2006.

[SW07] Daniel Sandler and Dan S. Wallach. Casting votes in the auditorium. In *EVT'07: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2007 on Electronic Voting Technology Workshop*, Berkeley, CA, USA, august 2007. USENIX Association.

[SW08] Daniel R. Sandler and Dan S. Wallach. The case for networked remote voting precincts. In *EVT'08: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2008 on Electronic Voting Technology Workshop*, Berkeley, CA, USA, july 2008. USENIX Association.

[TCL05] C. Tien, S. Chu, and Y. Lin. Four phases to construct problem-based learning instruction materials. In *PBL In Context Ð Bridging work and Education*, pages 117–133. Tampere University Press, 2005.

[TR00] Nicolaus Tideman and Daniel Richardson. Better voting methods through technology: The refinement-manageability trade-off in the single transferable vote. *Public Choice*, 103(1):13–34, April 2000.

[TRN08] Vanessa Teague, Kim Ramchen, and Lee Naish. Coercion-resistant tallying for STV voting. In *EVT'08: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2008 on Electronic Voting Technology Workshop*, Berkeley, CA, USA, july 2008. USENIX Association.

[TS02] L. Torp and S. Sage. *Problems as Possibilities: Problem-Based Learning for KŰ16 Education*. Association for Supervision and Curriculum Development (ASCD),, Alexandria, VA, USA, 2002.

[vGBS01] Jilles van Gurp, Jan Bosch, and Mikael Svahnberg. On the notion of variability in software product lines. In *WICSA*, pages 45–54. IEEE Computer Society, 2001.

[Win00] Jeannette M. Wing. Invited talk: Weaving formal methods into the undergraduate computer science curriculum. In Teodor Rus, editor, *Algebraic Methodology and Software Technology. 8th International Conference, AMAST 2000*, volume 1816 of *Lecture Notes in Computer Science*, pages 2–9, Iowa City, Iowa, USA, 2000. Springer.

[Win06] Jeannette M. Wing. Computational thinking. *Commun. ACM*, 49(3):33–35, 2006.

[Wit96] James Withey. Investment analysis of software assets for product lines. Technical Report CMU/SEI-96-TR-010, Software Engineering Institute, Carnegie Mellon University, November 1996.

[WK04] Brit J. Williams and Merle S. King. Implementing voting systems: the georgia method. *Commun. ACM*, 47(10):39–42, 2004.

[WKV10] Komminist Weldemariam, Richard A. Kemmerer, and Adolfo Villafiorita. Formal specification and analysis of an e-voting system. In *ARES*, pages 164–171. IEEE Computer Society, 2010.

[Woo96] D. R. Woods. *Problem-based Learning: how to gain the most from PBL.* Waterdown, Ontario, 1996.

[WV08] Komminist Weldemariam and Adolfo Villafiorita. Modeling and analysis of procedural security in (e)voting: The trentino's approach and experiences. In *EVT'08: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2008 on Electronic Voting Technology Workshop*, Berkeley, CA, USA, july 2008. USENIX Association.

[XM04a] Alexandros Xenakis and Ann Macintosh. Procedural security analysis of electronic voting. In Marijn Janssen, Henk G. Sol, and René W. Wagenaar, editors, *ICEC*, volume 60 of *ACM International Conference Proceeding Series*, pages 541–546. ACM, 2004.

[XM04b] Alexandros Xenakis and Ann Macintosh. Procedural security in electronic voting. In *HICSS*, 2004.

[XM05a] A. Xenakis and A. Macintosh. E-electoral administration: organizational lessons learned from the deployment of e-voting in the UK. In *Proceedings of the 2005 national conference on Digital government research*, pages 191–197. Digital Government Society of North America, 2005.

[XM05b] Alexandros Xenakis and Ann Macintosh. Procedural security and social acceptance in e-voting. In *HICSS*. IEEE Computer Society, 2005.

[XSHT08] Zhe Xia, Steve A. Schneider, James Heather, and Jacques Traoré. Analysis, improvement, and simplification of prêt à voter with paillier encryption. In *EVT'08: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2008 on Electronic Voting Technology Workshop*, Berkeley, CA, USA, july 2008. USENIX Association.

[YA07] Ahlem Ben Younes and Leila Jemni Ben Ayed. Using UML activity diagrams and Event-B for distributed and parallel applications. In *COMPSAC '07: Proceedings of the 31st Annual International Computer Software and Applications Conference - Vol. 1- (COMPSAC 2007)*, pages 163–170, Washington, DC, USA, 2007. IEEE Computer Society.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164. IEEE, 1982.

[YB08] Alec Yasinsac and Matt Bishop. The dynamics of counting and recounting votes. *IEEE Security and Privacy*, 6(3):22–29, 2008.

[Yee07a] Ka-Ping Yee. *Building Reliable Voting Machine Software*. PhD thesis, University of California, Berkeley, CA, USA, 2007.

[Yee07b] Ka-Ping Yee. Extending prerendered-interface voting software to support accessibility and other ballot features. In *EVT'07: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2007 on Electronic Voting Technology Workshop*, Berkeley, CA, USA, august 2007. USENIX Association.

[YWHB06] Ka-Ping Yee, David Wagner, Marti Hearst, and Steven M. Bellovin. Prerendered user interfaces for higher-assurance electronic voting. In *EVT'06: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, pages 6–6, Berkeley, CA, USA, 2006. USENIX Association.

[ZA08] André Zúquete and Filipe Almeida. Verifiable anonymous vote submission. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 2159–2166, New York, NY, USA, 2008. ACM.

[ZHJ03] Tewfik Ziadi, Loïc Hélouët, and Jean-Marc Jézéquel. Modélisation de lignes de produits en uml. *L'OBJET*, 9(1-2):227–240, 2003.

[ZS05] Bruno Zelic and Bernd Carsten Stahl. Does ontology influence technological projects? the case of irish electronic voting. In Klaus-Dieter Althoff, Andreas Dengel, Ralph Bergmann, Markus Nick, and Thomas Roth-Berghofer, editors, *Wissensmanagement (LNCS Volume)*, volume 3782 of *Lecture Notes in Computer Science*, pages 657–667. Springer, 2005.

5 CONCLUSIONS: WHAT HOLDS THE FUTURE? CONCLUSION: QUE NOUS PRESAGE LE FUTUR ?

I hope you become comfortable with the use of logic without being deceived into concluding that logic will inevitably lead you to the correct conclusion.
Neil Armstrong

Le présent est indéfini, le futur n'a de réalité qu'en tant qu'espoir présent, le passé n'a de réalité qu'en tant que souvenir présent.
Jorge Luis Borges

5.1 Software Engineering: things can only get better? — Génie logiciel : les choses peuvent-elles aller mieux ?

Twenty years ago — when I completed my thesis — I was sure that my (and other people's) research on formal object oriented development of software systems would become the de-facto industrial standard within a few years.

I was frustrated to learn — from fellow graduates who had moved on to industry — that poor software development practices were ubiquitous and that rigorous (never mind formal) methods were the exception rather than the rule.

Twenty years later, and my own experience from analysis of software systems suggests that not much has changed. Industrial pressure has tended to drive CS/CSSE programmes towards teaching layers of technology rather than fundamentals. Time spent talking to experienced "architects" suggests that they have yet to master techniques and tools that have existed for decades — version control, automated testing, configuration management, model driven development, etc .... They spend their time jumping from technology to technology that the foundational software engineering material is never mastered.

However, I am more upbeat than ever. Many of the current international software engineering research projects at European level have had to incorporate formal modelling. Multinational companies — like Microsoft, Intel, Google, etc ... — have invested in formal software engineering tool developmment. Without formal methods there

Il y a vingt ans — lorsque nous avonsachevé notre thèse — nous étions certain que notre recherche (et celle des autres) sur le développement formel OO de systèmes logiciels deviendrait le standard industriel de facto en quelques années.

Nous étions frustré d'apprendre — de la part d'autres diplômés qui étaient partis dans l'industrie — que de mauvaises pratiques dans le développement de logiciels étaient omniprésentes et que les méthodes rigoureuses (sans parler de formelles) étaient l'exception qui confirme la règle.

Vingt ans plus tard, et notre propre expérience tirée de l'analyse de systèmes de logiciels, suggère que peu a changé. La pression industrielle a eu tendance à conduire les programmes informatiques/génie logiciel vers l'enseignement de couches de technologie plutôt que les fondamentaux. Le temps passé à parler avec des architectes expérimentés suggère qu'ils ont encore à maîtriser les techniques et les outils qui ont existé depuis des décennies : contrôle de version, test automatisé, gestion de configuration, *model driven development*, etc... Il passent leur temps à sauter d'une technologie à l'autre de sorte que le fondement matériel du génie logiciel n'est jamais maîtrisé.

Pourtant, nous sommes plus optimiste que jamais. Actuellement, de nombreux projets internationaux de recherche en génie logiciel, au niveau européen, ont eu à incorporer la modélisation formelle. Des entreprises multinationales — telles que Microsoft, Intel, Google, etc... — ont investi dans le développement d'outil de

would be little or no advances in self* systems, automated service composition, aspect-oriented development, high performance computing, SPLs, etc...

For the first time, students are not just being presented with a stand-alone (often optional) formal methods course. They are seeing formal methods as another tool in their conceptual toolbox. Integrated development environments,such as Eclipse, have led students to quite naturally work with multiple models through a common interface. It is not so strange to see a student working with Java, JUnit, UML and Event-B views all competing for screen space at the same time. Software engineering can only get better when our students can work at different levels of abstraction and understand the underlying mathematics that links these levels.

5.2 Future Research Plans — Plans pour une recherche future

As well as the research proposed in the previous section, I have long term research plans that include:

Developing a software process to help small companies to adopt formal methods in an incremental fashion.

Developing a PBL software engineering programme where formal modelling is used in every module.

Analysing various styles of specification in Event-B in order to better understand the way in which engineers combine top-down and bottom-up development methods.

Twenty years ago I would never have thought that I would be researching voting systems. Twenty years from now who knows what I'll be doing, but I do know that — provided I am still working — my experience in formal software engineering will

génie logiciel formel. Sans les méthodes formelles, il n'y aurait peu ou pas d'avancées dans les secteurs de recherche offrant un challenge au domaine du logiciel : *"self* systems"*, composition de service automatisé, *aspect-oriented development*, computation à haute performance, SPLs, etc...

Pour la première fois, on ne présente pas aux étudiants simplement un cours autonome (et souvent optionnel) de méthodes formelles. Ils voient les méthodes formelles comme un autre outil dans leur boîte à outils conceptuelle. Des environnements de développement intégrés, tels que Eclipse, ont conduit les étudiantsà travailler assez naturellement avec de multiples modèles, à travers une interface commune. Il n'est pas exceptionnel de voir un étudiant travailler avec Java, JUnit, UML et sur des vues RODIN/ Event-B, rivalisant tous pour une espace sur l'écran en même temps. Le génie logiciel peut seulement s'améliorer quand nos étudiants peuvent travailler à différents niveaux d'abstraction et comprendre les mathématiques sous-jacents qui relient ces niveaux.

Tout autant que la recherche proposée dans la section précédente, nous avons des plans de recherche à long terme qui incluent :

La construction d'un processus de développement logiciel aidant les petites entreprises à adopter des méthodes formelles d'une manière incrémentielle.

Le développement d'un programme de génie logiciel PBL où la modélisation formelle est utilisée dans chaque module.

L'analyse de styles variés de specification dans Event-B afin de mieux comprendre la façon dont les ingénieurs combinent les méthodes de développement de haut en bas et de bas en haut.

Il y a vingt ans, nous n'aurions jamais imaginé faire de la recherche dans le domaine des systèmes de vote. Dans vingt ans, qui sait ce que nous ferons, mais nous savons une chose — à condition de toujours travailler — c'est qu'il existe a de fortes chances pour que notre

most likely help me in doing my work well.

expérience dans le génie logiciel formel nous aiderons à bien faire notre travail.