Dr. David J. Pearce

Personal Details

Phone	+64 (0)4 463 5833 (office) +64 (0)21 037 4371 (mobile)
Email	david.pearce@ecs.vuw.ac.nz
Websites	http://www.ecs.vuw.ac.nz/~djp http://whiley.org

Employment

2013 –	Programme Director for Software Engineering Major in Bachelor of Engineering, Victoria University of Wellington, NZ.
2009 –	Senior Lecturer in Computer Science (Full Time), Victoria University of Wellington, NZ.
2004 – 2009	Lecturer in Computer Science (Full Time), Victoria University of Wellington, NZ.
2004	Three month internship at IBM Hursley, UK, working under Robert Berry on AspectJ.
2004	Three months as Research Assistant, Imperial College London, UK
2002	Three month internship at Bell Labs, New Jersey, working under Oskar Mencer on a compiler for Field-Programmable Gate Arrays.
1999	Six months at Philips Research Laboratories, Redhill, UK, working under Johnny Farringdon on devices for wearable computing.

Education

2000 – 2005	PhD, *"Some Directed Graph Algorithms and their Application to Pointer Analysis"*, supervised by Prof. Paul Kelly, Imperial College London.
1996 – 2000	MEng in Computing (1st class) from Imperial College London. My final year project lead to a paper presented at the TOOLS 2002 conference.

Awards

2010	Science, Engineering and Architecture & Design Excellence Award, Victoria University of Wellington
2009	Early Career Research Award, Victoria University of Wellington.
2005	PhD thesis nominated by external examiners — Mark Harman and Andy King — for British Computer Society (BCS) doctoral dissertation prize.

Referees

Prof. James Noble	School of Engineering and Computer Science, Victoria University of Wellington, PO Box 600, Wellington, NZ. Email: kjx@ecs.vuw.ac.nz
Dr. Ian Welch	School of Engineering and Computer Science, Victoria University of Wellington, PO Box 600, Wellington, NZ. Email: ian.welch@ecs.vuw.ac.nz

Research

Research Statement

Software is ubiquitous in everyday life, and underpins much of modern society. Two questions for anyone wishing to develop software are: *How do we solve this problem with software?* and, *how do we know the software works correctly?* My research follows naturally from these, and focuses on the development and application of high-quality software. More specifically, I am interested in three sub-fields relating to software development:

Verifying Compilers. In 2003, Prof. Sir Tony Hoare (ACM Turing Award Winner, FRS) identified the creation of a *verifying compiler* as a grand challenge for computer science. A verifying compiler "*uses automated mathematical and logical reasoning to check the correctness of the programs that it compiles*". In other words, software developed using a verifying compiler will have significantly fewer errors than if developed by traditional means. My main achievement in this field is:

1. **Developing a verifying compiler system called *Whiley***. This has involved collaboration with academics from Victoria University and Waikato University and has attracted Marsden funding. The system has been used in teaching the course SWEN224 "Formal Foundations of Programming" for the past two years, and as the subject of eight final-year projects. In 2013, the project was featured in the *Dominion Post*. It has received some international recognition and been noted for influencing at least one widely-used language (Ceylon). Since 2010, my whiley.org website has received over 150K unique visitors who generated over 390K page views.

Algorithms. The study of algorithms is really about determining good solutions to a problem under given constraints and, as such, underpins much of computer science. My main achievements in this field are:

2. **Developing a tool for computing Tutte Polynomials, which remains the most efficient tool available.** Tutte Polynomials are of significant interest in mathematics, statistical physics, and biology (e.g. for analysing knotted strands of DNA). Our algorithm is the most efficient available and enabled us to compute, for the first time, the Tutte Polynomial of the Truncated Icosahedron — the largest graph for which this has been done. Additionally, we found a counterexample to a conjecture by Welsh that had remained unsolved for 25 years. Numerous researchers around the globe have downloaded and used our system. For example, Dr Frank Emmert-Streib from Queens University Belfast, explored its use in protein analysis. Finally, our algorithm has recently been incorporated into the widely used Mathematica application (version 10.0).

3. **Developing several algorithms for maintaining the topological order of a directed graph under edge insertions**. These algorithms have many applications and have been used (by others) for software verification, machine-learning, sequence alignment of proteins, and more.

4. **Developed an improved algorithm for topologically sorting directed graphs.** This improves the space requirements over Tarjan's well-known algorithm, and has been independently integrated into the widely-used SciPy library for scientific computing, specifically because of its ability to handle larger graphs in practice.

Program Analysis. This is the study of tools and techniques for automatically analysing and understanding software. There are numerous applications of this, including *software verification*, *compiler optimisation* and *program understanding*. My main achievement in this field is:

5. **Developing a novel analysis for software written in the C programming language**. This has been incorporated into the GNU C Compiler (GCC) v4.1.0 — the most widely used compiler in the world; as such it is used by hundreds of thousands of programmers every day.

Contribution to Research

2015 Judge for Prime Minister's MacDiarmid Emerging Scientist Prize (RSNZ)
2015 Guest editor for Special Issue of SLE'14 in *Computer Languages, Systems and Structures* (an Elsevier Journal)
2014 Program Co-chair for *Conference on Software Language Engineering (SLE'14)*,Vasteras, Sweden
2014 Program Chair for *Workshop on Formal Techniques for Java-like Programs (FTfJP'14)*, Uppsala, Sweden
2014- Steering Committee Member for *Conference on Software Language Engineering (SLE)*
2014 Judge for Prime Minister's MacDiarmid Emerging Scientist Prize (RSNZ)
2013- Editor of *Science of Computer Programming* (an Elsevier Journal, CoRE journal ranking: A)
2013 Judge for Prime Minister's MacDiarmid Emerging Scientist Prize (RSNZ)

Since 2004, I have served as an external examiner for four PhD and four Masters Theses, and have reviewed over 150 papers for international conferences and journals, many of which are highly ranked in my field. These include: ACM POPL, ACM PLDI, ACM TOPLAS, ACM OOPSLA, IEEE TSE, ECOOP, ESA, CC, ESOP, Elsevier SCP, Wiley SP&E, and more.

Program Committee Service

ACM/SPEC ICPE'13 & '14, ASWEC'14, BEAT'14, ACM PLDI'13 (ERC member), ACM HILT'14, ACM SAC'13 (OOPS Track), HILT'13, ASWEC'13, PLACES'12, ACM SAC'12 (OOPS Track), ASWEC'10, ACM SAC'10 (PCS Track), TAIC-PART'09, ACM SAC'09 (PCS Track), RAOOL'09, RAOOL'08, ACM SAC'08 (PCS Track), CATS'08, CATS'07

Organising Committee Service

2010 Dagstuhl Seminar Organiser, *"Relationships, Objects, Roles and Queries in Modern Programming Languages"*, Seminar #10152, Dagstuhl, Germany

2010 *Matroid Computation Meeting*, Victoria University, Wellington, NZ

2010 *Australasian Computing Doctoral Consortium (ACDC)*, Brisbane, Australia

2009 *Australasian Computer Science Week (ACSW)*, Wellington, NZ

2009 *Australasian Computing Doctoral Consortium (ACDC)*, Wellington, NZ

Grants

RSNZ **Fast Start Marsden Grant (Primary Investigator)** entitled "Attacking the Verifying Compiler Grand Challenge", $345K. 1st Mar 2012 — 1st Mar 2015. Royal Society of New Zealand. 88 / 1078 applications were successful in that round (8.2%)

VUW SRG Science Faculty Research Grant, "Immutability and Ownership Inference for Java", $6000, 2009.

VUW SRG Science Faculty Research Grant, "Runtime Specification Checking using JKit", $6000, 2009.

BuildIT Emerging Research Travel Grant, $3850, BuildIT, 2008.

VUW SRG Science Faculty Research Grant, "Programming Languages for Verifying Compilers", $6000, 2008.

VUW URF University Research Fund, "Qualified Type Inference for Java", $8000, grant number 1620-26218-1426, 2006.

VUW URF University Research Fund, "Finding null-pointer errors in Java programs", $17000, grant number 1620-24443-1387, 2005.

Teaching

Teaching Approach

In my time as a lecturer, through observation of others and comments from students, I have learned a number of things about how to teach effectively:

1. **Clarity and delivery of presentation is crucial**. If students find lectures monotonous and boring, or — worse still — cannot understand what is being discussed, they will switch off and, eventually, stop attending. To combat this, I work hard during my lectures to appear animated and enthusiastic, and to present topics in clear and simple ways. I often break up my lectures with short quizzes, where students work in pairs on a short problem involving the material being discussed. Many students have commented on how they find this helps them learn.

2. **Maintaining the right attitude towards students is crucial**. Students learn better if they feel the "system" wants to help, rather than hinder them. To that end, I always treat student questions in class with the utmost respect. I also always maintain an open door policy, and find time to talk with students whenever possible. I genuinely believe this makes a difference, especially to those weaker students who are struggling.

3. **Good feedback is essential**. Providing good feedback on assignments, labs and mid-term tests is critical to student learning. At the same time, providing timely feedback is difficult, particularly for large courses. To reduce turn-around time, I make extensive use of online systems — many developed by myself — to improve efficiency. As an example, my student code submissions are automatically run against test inputs, and marked against expected output. Students are then emailed with results detailing which tests failed and why.

Overall, I have enjoyed my time as a lecturer so far and believe that I am able to help students learn, understand and — most importantly — enjoy the subjects I am teaching.

(See my Teaching Performance Profile on the last page)

Teaching Commendations

Victoria University provides a mechanism for conducting teaching evaluations, and I have included several of my evaluation summaries as evidence of my teaching quality. Students can also provide written comments, and I always take time to look through these. A few selected comments from my recent evaluations are:

Please list 2 or 3 specific things that this teacher did that stimulated and/or helped you to learn.
- *His attention to each and every student when his attention could be spared*
- *His patience with difficult people in difficult situations*
- *Great lecture slides*
- *Involved the students*
- *Explained everything in an easy to comprehend manner & in great detail for the harder stuff.*
- *Fun lecturing style*
- *Enthusiastic*
- *Made sure everyone understood the topic*
- *Happy to explain concepts one-on-one or whenever*

Thesis Supervision

PhD	Timothy Jones (completion expected 2016, I am Secondary Supervisor)
PhD	Michael Homer (completed 2014, I was Secondary Supervisor)
PhD	Roman Klaplaukh (completed 2014, I was Principal Supervisor)
PhD	Art Protin (incomplete, I was Principal Supervisor)
PhD	Stephen Nelson (completed 2012, I was Principal Supervisor)
MSc	Paran Haslett (completed 2014, I was Principal Supervisor)
MSc	Constantine Dymnikov (completed 2011, I was Secondary Supervisor)
MSc	Chris Male (completed 2009, I was Principal Supervisor)
MSc	Darren Willis (completed 2008, I was Secondary Supervisor)

Teaching Experience

2015 SWEN430 "*Compiler Engineering*" (≈20 students, co-taught, coordinator)
SWEN301 "*Software Engineering*" (≈50 students, co-taught)
SWEN221 "*Software Development*" (≈200 students, co-taught, coordinator)
SWEN222 "*Software Design*" (≈140 students, co-taught, coordinator)
SWEN224 "*Software Design*" (≈140 students, co-taught, coordinator)

2014 SWEN430 "*Compiler Engineering*" (≈20 students, co-taught)
SWEN221 "*Software Development*" (≈170 students, co-taught, coordinator)
SWEN222 "*Software Design*" (≈120 students, co-taught, coordinator)

2013 SWEN430 "*Compiler Engineering*" (≈20 students, co-taught, coordinator)
SWEN221 "*Software Development*" (≈150 students, co-taught, coordinator)
SWEN222 "*Software Design*" (≈100 students, co-taught)
SWEN102 "*Introduction to Software Modelling*" (≈150 students, co-taught, coordinator)

2012 SWEN221 "*Software Development*" (≈150 students, co-taught, coordinator)
ENGR101 "*Engineering Technology*" (≈150 students, co-taught)
(Reduced load due to Sabbatical)

2011 SWEN430 "*Compiler Engineering*" (≈20 students, co-taught, coordinator)
SWEN221 "*Software Development*" (≈120 students, co-taught, coordinator)
SWEN222 "*Software Design*" (≈85 students, co-taught)
SWEN102 "*Intro to Software Modelling*" (≈100 students, co-taught, coordinator)
ENGR101 "*Engineering Technology*" (≈150 students, co-taught)
ENGR489 "*Final Year Project*" (≈20 students, co-taught, coordinator)

2010 ENGR489 "*Final Year Project*" (≈20 students, co-taught, coordinator)
SWEN430 "*Compiler Engineering*" (≈20 students, co-taught)
SWEN221 "*Software Development*" (≈100 students, co-taught, coordinator)
SWEN222 "*Software Design*" (≈70 students, co-taught, coordinator)
SWEN102 "*Intro to Software Modelling*" (≈100 students, co-taught)

2009 COMP431 "*Compilers*" (≈10 students, co-taught, coordinator)
SWEN102 "*Intro to Software Modelling*" (≈70 students, co-taught, coordinator)
(Reduced load due to Sabbatical)

2008 COMP431 "*Compilers*" (≈10 students, co-taught)
COMP205 "*Software Design and Engineering*" (≈80 students, co-taught, coordinator)
COMP202 "*Formal Methods of Computer Science*" (≈30 students, co-taught)
SWEN102 "*Intro to Software Modelling*" (≈70 students, co-taught)

2007 COMP471 "*Compilers*" (≈10 students, co-taught, coordinator)
COMP205 "*Software Design and Engineering*" (≈80 students, co-taught, coordinator)
SWEN102 "*Intro to Software Modelling*" (≈50 students, co-taught)

2006 COMP463 "*Advanced Software Engineering*" (≈20 students, coordinator)
COMP205 "*Software Design and Engineering*" (≈80 students, co-taught)

2005 COMP463 "*Advanced Software Engineering*" (≈20 students, coordinator)
COMP205 "*Software Design and Engineering*" (≈80 students, co-taught)

2004 COMP203 "*Computer Organisation*" (≈100 students, co-taught)

Publications

(*See my publications profile:* http://scholar.google.co.nz/citations?user=x2QxevkAAAAJ)

Refereed Journal Articles

1. **David J. Pearce** and Lindsay Groves. Designing a Verifying Compiler: Lessons Learned from Developing Whiley. In *Science of Computer Programming*, (to appear), 2015. **CoRE**[1] **journal ranking: A**
2. **David J. Pearce**. A Space Efficient Algorithm for Detecting Strongly Connected Components. In *Information Processing Letters*, 116(1):47–52, 2015. doi:10.1016/j.ipl.2015.08.010
3. Chris Male, **David J. Pearce**, Alex Potanin and Constantine Dymnikov, Formalisation and Implementation of an Algorithm for Bytecode Verification of @NonNull Types. In *Science of Computer Programming (SCP)*, 76(7):587–608, 2011. **(CoRE journal ranking: A)** DOI: 10.1016/j.scico.2010.10.004
4. Gary Haggard, **David J. Pearce** and Gordon Royle, Computing Tutte Polynomials. In *ACM Transactions on Mathematical Software (TOMS)*, article 24, 2010. **(CoRE journal ranking: A*, citations: 32)** DOI: 10.1145/1824801.1824802
5. **David J. Pearce**, Gary Haggard and Gordon Royle, Edge-Selection Heuristics for Computing Tutte Polynomials. In *Chicago Journal of Theoretical Computer Science*, Article 6, 2010. DOI: 10.4086/cjtcs.2010.006
6. **David J. Pearce**, Paul H.J. Kelly and Chris Hankin, Efficient Field-Sensitive Pointer Analysis for C. In *ACM Transactions on Programming Languages and Systems (TOPLAS)*, volume 30(1), 2007. ACM Press. **(CoRE journal ranking: A*, citations: 129)** DOI: 10.1145/1290520.1290524
7. **David J. Pearce**, Matthew Webster, Robert Berry and Paul H.J. Kelly, Profiling with AspectJ. In *Software: Practice and Experience*, Volume 37(7), pages 747–777, 2007, Wiley. **(CoRE journal ranking: A, citations: 58)** DOI: 10.1002/spe.788
8. **David J. Pearce** and Paul H.J. Kelly, A Dynamic Topological Sort Algorithm for Directed Acyclic Graphs. In *ACM Journal of Experimental Algorithmics (JEA)*, volume 11, 2007. ACM Digital Library. **(CoRE journal ranking: A, citations: 55)** DOI: 10.1145/1187436.1210590
9. **David J. Pearce**, Paul H.J. Kelly and Chris Hankin. Online Cycle Detection and Difference Propagation: Applications to Pointer Analysis. In *Software Quality Journal*, volume 12(4):309-335, 2004, Kluwer Academic Publishers. **(citations: 34)** DOI: 10.1023/B:SQJO.0000039791.93071.a2

Refereed Conference Papers

10. Roma Klapaukh, **David J. Pearce** and Stuart Marshall. Comparing Graph Layouts for Vertex Selection Tasks. In *Proceedings of the Australian Conference on Human Computer Interaction (OzCHI)*, (to appear), 2015.
11. **David J. Pearce**. The Whiley Rewrite Language (WyRL). In *Proceedings of the Conference on Software Language Engineering (SLE)*, (to appear), 2015.
12. Roma Klapaukh, **David J. Pearce** and Stuart Marshall. Towards a Vertex and Edge Label Aware Force Directed Layout Algorithm. In *Proceedings of the Australasian Computer Science Conference (ACSC)*, pages 29–37, 2014.
13. **David J. Pearce** and Lindsay Groves. Whiley: a Platform for Research in Software Verification. In *Proceedings of the Conference on Software Language Engineering (SLE)*, volume 8225 of LNCS, pp238–248, 2013. DOI: 10.1007/978-3-319-02654-1_13
14. Constantine Dymnikov, **David J. Pearce** and Alex Potanin. OwnKit: Inferring Modularly Checkable Ownership Annotations for Java. In Proceedings of the Australasian Software Engineering Conference (ASWEC), pages 181–190, 2013. DOI: 10.1109/ASWEC.2013.30
15. **David J. Pearce**. Sound and Complete Flow Typing with Unions, Intersections and Negations. In *Proceedings of the Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 7737 of Lecture Notes in Computer Science, pages 335–354, 2013. DOI: 10.1007/978-3-642-35873-9_21
16. Michael Homer, James Noble, Kim B. Bruce, Andrew P. Black and **David J. Pearce**. Patterns as Objects in Grace. In *Proceedings of the Dynamic Languages Symposium (DLS)*, pages 17–28, 2012. DOI: 10.1145/2384577.2384581.

[1] Note, CoRE is the Computing Research and Education Association of Australia

17. Stephen F. Nelson, **David J. Pearce** and James Noble. Profiling Field Initialization for Java. In *Proceedings of the Conference on Runtime Verification (RV)*, volume 7687 of Lecture Notes in Computer Science, pages 292–307, 2012. DOI: 10.1007/978-3-642-35632-2_28.
18. **David J. Pearce**, JPure: a Modular Purity System for Java. In *Proceedings of the Conference on Compiler Construction (CC)*, volume 6601 of LNCS, pages 104–123, 2011. DOI: 10.1007/978-3-642-19861-8_7
19. Stephen F. Nelson, **David J. Pearce** and James Noble. Understanding the Impact of Collection Contracts on Design. In *Proceedings of the Conference on Objects, Models, Components, Patterns (TOOLS EUROPE)*, pages 61–78, 2010. DOI: 10.1007/978-3-642-13953-6_4
20. **David J. Pearce** and Paul H.J. Kelly, A Batch Algorithm for Maintaining a Topological Order. In *Proceedings of the Australasian Computer Science Conference (ACSC)*, pages 79–88, 2010.
21. **David J. Pearce**, Gary Haggard and Gordon Royle, Edge-Selection Heuristics for Computing Tutte Polynomials. In *Proceedings of the Computing: The Australasian Theory Symposium (CATS)*, pages 153–162, 2009.
22. Darren Willis, **David J. Pearce** and James Noble, Caching and Incrementalisation for the Java Query Language. In *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages & Applications (OOPSLA)*, pages 1–17, 2008. **(CoRE conference ranking: A*, citations: 38)** DOI: 10.1145/1449955.1449766.
23. **David J. Pearce** and James Noble, Patterns for ADT Optimisation. In *Proceedings of the conference on Pattern Languages of Programs (PLoP)*, Article 26, 2008. DOI: 10.1145/1753196.1753227
24. Chris Male, **David J. Pearce**, Alex Potanin and Constantine Dymnikov, Java Bytecode Verification for @NonNull Types. In *Proceedings of the Conference on Compiler Construction (CC)*, pages 229–244, 2008. Springer-Verlag. **(CoRE conference ranking: A, citations: 28)** DOI: 10.1007/978-3-540-78791-4_16
25. Bennett Thompson, **David J. Pearce**, Craig Anslow and Gary Haggard. In *Proceedings of ACM Symposium on Software Visualisation (SoftViz)*, pages 211–212, 2008.
26. James Noble, Arno Schmidmeier, **David J. Pearce** and Andrew Black, Patterns of Aspect-Oriented Design. In *Proceedings of the European Conference on Pattern Languages of Programs (EuroPLOP)*, pages 769–796, 2007.
27. **David J. Pearce**, Bennett Thompson and Gary Haggard, Visualising the Tutte Polynomial Computation. In *Proceedings of the New Zealand Conference on Software Engineering (SIENZ)*, 2007.
28. **David J. Pearce** and James Noble. Relationship Aspects. In *Proceedings of the ACM Conference on Aspect-Oriented Software Development (AOSD)*, pages 75–86, 2006. ACM Press. **(CoRE conference ranking: A, citations: 84)** DOI: 10.1145/1119655.1119668
29. Darren Willis, **David J. Pearce** and James Noble, Efficient Object Querying for Java. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, pages 28–49, 2006, Springer-Verlag. **(CoRE conference ranking: A, citations: 57)** DOI: 10.1007/11785477_3
30. **David J. Pearce** and James Noble, Relationship Aspect Patterns. In *Proceedings of the European Conference on Pattern Languages of Programs (EuroPLOP)*, pages 531–546, 2006. Hillside Publishers.
31. James Noble, Robert Biddle, Ewan Tempero and **David J. Pearce**, Towards a Semiotics of Object- and Aspect-Oriented Design. In *Proceedings of the European Conference on Computing and Philosophy (ECAP)*, 2006.
32. Dong-U Lee, Oskar Mencer, **David J. Pearce** and Wayne Luk. Automating Optimized Table-with-Polynomial Function Evaluation for FPGAs. In *Proceedings of conference on Field-Programmable Logic and its Applications (FPL04)*, LNCS v3203, pages 364–373, 2004. Springer-Verlag. DOI: 10.1007/978-3-540-30117-2_38
33. Oskar Mencer, **David J. Pearce**, Lee W. Howes and Wayne Luk, Design Space Exploration with A Stream Compiler, In *Proceedings of IEEE Conference on Field-Programmable Technology (FPT)*, pages 270–277, 2003. **(citations: 45)** DOI: 10.1109/FPT.2003.1275757
34. **David J. Pearce**, Paul H.J. Kelly, Tony Field and Uli Harder. GILK: A Dynamic Instrumentation Tool for the Linux Kernel. In *Proceedings of the conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS)*, volume 2324 of Lecture Notes in Computer Science, pages 220–226, 2002. Springer-Verlag. DOI: 10.1007/3-540-46029-2_16 **(citations: 41)**

Refereed Workshop Papers

35. **David J. Pearce**. Some Usability Hypotheses for Verification. In *Proceedings of the Workshop on Evaluation and Usability of Programming Languages (PLATEAU)*, (to appear), 2015.

36. **David J. Pearce**. Integer Range Analysis for Whiley on Embedded Systems. In *Proceedings of the IEEE/IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems*, pages 26–33, 2015. http://dx.doi.org/10.1109/ISORCW.2015.54
37. **David J. Pearce** and Lindsay Groves. Reflections on Verifying Software with Whiley. In *Proceedings of the Workshop on Formal Techniques for Safety-Critical Systems (FTSCS)*, Communications in Computer and Information Systems (CCIS), 419:142–159, 2014.
38. **David J. Pearce**. A Calculus for Constraint-Based Flow Typing. In *Proceedings of the Workshop on Formal Techniques for Java-like Languages (FTFJP)*, Article 7, 2013. DOI: 10.1145/2489804.2489810
39. Marco Servetto, **David J. Pearce**, Lindsay Groves, and Alex Potanin. Balloon Types for Safe Parallelisation over Arbitrary Object Graphs. In Proceedings of the Workshop on Determinism and Correctness in Parallel Programming, 2013.
40. **David J. Pearce** and James Noble. Implementing a Language with Flow-Sensitive and Structural Typing on the JVM. In *Proceedings of the Workshop on Bytecode Semantics, Verification, Analysis and Transformation (BYTECODE)*, ENTCS 279(1):47–59, 2011. DOI: 10.1016/j.entcs.2011.11.005
41. Stephen F. Nelson, **David J. Pearce** and James Noble, Implementing relationships using Affinity. In *Proceedings of the Workshop on Relationships and Associations in Object-Oriented Languages (RAOOL)*, pages 5–8, 2009. DOI: 10.1145/1562100.1562102
42. James Noble, **David J. Pearce** and Lindsay Groves, Introducing Software Modelling with Alloy at VUW. In *Proceedings of the Workshop on Formal Methods in Computer Science Education (FORMED)*, pages 81–90, 2008
43. Stephen F. Nelson, **David J. Pearce** and James Noble, Implementing First-Class Relationships in Java. In *Proceedings of Workshop on Relationships and Associations in Object-Oriented Languages (RAOOL)*, 2008.
44. Roshan Ramachandran, **David J. Pearce** and Ian Welch, AspectJ for Multilevel Security. In *Proceedings of the Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS)*, pages 13–17, 2006. **(citations: 30)**
45. **David J. Pearce**, Paul H. J. Kelly and Chris Hankin. Efficient Field-Sensitive Pointer Analysis for C, In *Proceedings of the ACM workshop on Program Analysis for Software Tools and Engineering (PASTE)*, pages 37–42, 2004. ACM Press. DOI: 10.1145/996821.996835
46. **David J. Pearce** and Paul H. J. Kelly. A Dynamic Algorithm for Topologically Sorting Directed Acyclic Graphs, In *Proceedings of the Workshop on Efficient and experimental Algorithms (WEA'04)*, LNCS v3059, pages 383–398, 2004. Springer-Verlag. **(citations: 29)** DOI: 10.1007/978-3-540-24838-5_29
47. **David J. Pearce**, Paul H. J. Kelly and Chris Hankin. Online Cycle Detection and Difference Propagation for Pointer Analysis. In *Proceedings of IEEE Workshop on Source Code Analysis and Manipulation (SCAM)*, pages 3–12, 2003. **(citations: 39)** DOI: 10.1109/SCAM.2003.1238026

Technical Reports

35. Roman Klapaukh, **David J. Pearce** and Stuart Marshall **An Algorithm for Measuring the Symmetry Exhibited by Graph Layouts**. Submitted to *Journal of Experimental Algorithmics*, 2015.
36. **David J. Pearce**. Practical Verification Condition Generation for a Bytecode Language. Victoria University of Wellington, Technical Report #ECSTR14-07, **2014**.
37. **David J. Pearce** and L. Groves. A Calculus for Verification Condition Generation. Victoria University of Wellington, Technical Report #ECSTR13-04, **2013**.
38. **David J. Pearce**. An Algorithmic Framework for Recursive Structural Types. Victoria University of Wellington, Technical Report #ECSTR11-10, **2011**.
39. **G. Haggard**, **David J. Pearce** and *G. Royle*. Computing Tutte Polynomials. Isaac Newton Institute for Mathematical Sciences, #NI09024-CSM, **2009**.

Victoria
UNIVERSITY OF WELLINGTON
Te Whare Wānanga
o te Ūpoko o te Ika a Māui

Centre for Academic Development

Teaching Performance Profile for David Pearce (3560)

Prepared by CAD
23 MAY 2014
Signed

Course	Year	Activity	Clarity of communication	Organisation of teaching	Stimulation of interest	Attitude towards students	Overall effectiveness	Number of respondents
(pre-2007)								

Course	Year/Trimester	Activity	Clarity of communication	Organisation of teaching	Stimulation of interest	Attitude towards students	Overall effectiveness	Number of respondents	Enrolled	Response rate	Number of hours taught
(2007 onwards)											
SWEN221	2010T1	teaching	1.4	1.7	1.4	1.1	1.5	61	99	62%	12
SWEN102	2010T2	teaching	1.1	1.5	1.4	1.0	1.2	49	121	40%	18
SWEN222	2010T2	teaching	1.3	1.3	1.6	1.1	1.2	36	67	54%	8
SWEN430	2010T2	teaching	1.2	1.4	1.9	1.1	1.4	14	20	70%	12
SWEN102	2011T2	teaching	2.0	2.1	2.1	1.1	2.3	56	109	51%	12
SWEN222	2011T2	teaching	1.2	1.3	1.3	1.1	1.1	46	74	62%	12
SWEN430	2011T2	teaching	1.8	1.2	1.8	1.0	1.2	4	11	36%	16
ENGR101	2012T1	teaching	1.7	1.9	1.7	1.1	1.7	133	172	77%	9
SWEN221	2012T1	teaching	1.2	1.6	1.8	1.1	1.3	63	120	53%	10
SWEN221	2013T1	teaching	1.2	1.3	1.5	1.1	1.3	108	159	68%	12
SWEN102	2013T2	teaching	1.3	1.5	1.9	1.1	1.6	97	208	47%	11
SWEN430	2013T2	teaching	1.2	1.4	1.2	1.1	1.2	7	13	54%	12
SWEN221	2014T1	teaching	1.2	1.3	1.5	1.0	1.3	91	158	58%	14
SWEN430	2014T1	teaching	1.1	1.1	1.7	1.0	1.1	5	12	42%	6

Teaching Performance Profile