PhD Research Proposal - Visualising Software Corpus Analysis

Craig Anslow
School of Mathematics, Statistics and Computer Science
Victoria University of Wellington
Email: craig@mcs.vuw.ac.nz

1 Summary

Despite the spread of software development and software usage, we have almost no dependable data on how software is actually written in practice. Understanding the shape of existing software is an important step to understanding what good software looks like.

Our proposal is to undertake quantitative studies of the way software is actually written in practice and evolved over time by collecting large corpora of software in object-oriented and aspect-oriented programming languages. We will then create tools to produce visualisations of the structure and behaviour of the software using visualisation techniques to characterize each language's characteristic patterns of usage and frequency. In other disciplines, e.g. applied linguistics [12], this kind of approach is well established.

Our proposed kind of corpus analysis will expose how programmers actually use languages, what features of languages are used, and better inform programming pedagogy, software language design, and software understanding. A key observation leading to this proposal is that software that could comprise such corpora have become available for study only in the last decade. One source of corpora will be free and open-source software (FOSS) that is freely accessible over the Internet.

2 Specific Objectives of Research

Our objectives are as follows:

- We want to expose how programmers actually write object-oriented software, use object-oriented languages, what features of languages are used, and how software evolves over time. The software corpus analysis will help to better inform programming pedagogy, software language design, and understand software.

- We aim to produce tools that can create effective visualisations of software to help determine the quality of software for maintenance purposes. Software maintenance is reported to be about 70% of the total cost of a software product [3, 24]. Understanding what causes these costs is an ongoing research problem in software engineering. Examining existing software to determine where the costs are is a good idea.

- We aim to analyse object-oriented programs, design patterns, and aspects from the software corpus using software visualisation techniques [20, 19]. Software visualisation [10, 25, 29, 7] is the use of the crafts of typography, graphic design, animation, and cinematography with modern human-computer interaction and computer graphics technology to facilitate both the human understanding and effective use of computer software.

- We want to see what effect design patterns [11, 27] have in object-oriented software development. Using design patterns can help speed up the software development process by providing tested, proven development paradigms. However if patterns are used inaccurately this can lead to inefficient solutions and incorrect implementations, hence requiring majoring refactoring.

- We want to see how applicable Aspect Oriented Programming (AOP) [17, 18, 13, 6] is used for the development of object-oriented software. AOP attempts to aid programmers in the separation of concerns, specifically crosscutting concerns, as an advance in modularisation and to reduce costs in the software development process. AOP does so using a combination of language changes, environment, and

methodology. However if there are few opportunities where AOP is applicable then the possible benefits of AOP are likely to be limited.

3 Related Work

Some work related to understanding software, and visualising software corpora, design patterns, and aspects include:

- The Lego Hypothesis [23, 26] says that software can be put together like Lego out of small interchangeable components. Software constructed according to this theory should show certain kinds of structure: components should be small and should only refer to a small number of closely related components. An in-depth study of the structure of Java programs collected 56 applications of varying sizes and measured their key structural attributes [1]. They found evidence that some relationships follow power-laws, while others do not.

- Noble and Biddle [21, 22] have used information visualisation techniques [5, 2, 28] to investigate the layouts and programming style of over 1,000 Nord Modular programs. They found that although modules could be positioned freely within a program, particular types of modules were generally found in stereotypical locations.

- Jerding [14] has looked at visualising design patterns in the execution of object oriented programs, while others have looked at visualising design patterns using UML [8], 3D [4] and web services [9].

- Khaled et al. [16] have used AspectJ to collect program monitoring information for visualising UML sequence diagrams of running programs and algorithm animations of sorting algorithms. They also created domain specific visualisations for a library system. Other work has looked at using AOP and Eclipse to help people learn object-oriented programming [15].

4 Research Methodology

1. Literature review on how other disciplines do corpus analysis, existing software corpus analysis, techniques for visualising software, open source software, and studies on how software is written.

2. Collect appropriate data from open source software repositories such as SourceForge.net, Free Software Foundation, and the Apache Open Source Foundation.

3. Compare the literature on how to write software versus how software is actually written. Identify the common features and differences between the theoretical and practitioner approaches.

4. Identify the problems and opportunities that exist with the way software is actually written and evolved versus the many methodologies, patterns and standards for designing and programming software.

5. Develop prototype tools to create visualisations of the structure and behaviour of the software in the corpus.

6. Evaluate the tools for creating software visualisations and the visualisations produced from the tools.

7. Analysis of the software corpus and visualisations. Characterize each language's characteristic patterns of usage. Identify the common features and differences between the languages. Identify and document how design patterns and aspects have been used.

8. Discussion of the results.

9. Report on the results by writing a thesis.

5 Milestones and Expected Timelines

Activity	**Month**	**Year**
Literature Review	March-September	2008
Collect Data	June-November	
Full PhD Proposal	November-December	
PhD Proposal Seminar	February	2009
Design of visualisation system	March-April	
Implementation of system	April-September	
Evaluate and test system	September-October	
Create visualisations	October-December	
Analysis of results	February-March	2010
Write up thesis	April-September	
Draft 1 Thesis	October	
Draft 2 Thesis	December	
Final Thesis	January	2011
Submit Thesis	February	
Thesis Seminar	March	

References

[1] Gareth Baxter, Marcus Frean, James Noble, Mark Rickerby, Hayden Smith, Matt Visser, Hayden Melton, and Ewan Tempero. Understanding the shape of java software. In *OOPSLA '06: Proceedings of the 21st ACM SIGPLAN conference on Object-oriented Programming, Languages, Systems and Applications*. ACM Press, October 2006.

[2] Ben Shneiderman Benjamin B. Bederson, editor. *The Craft of Information Visualization: Readings and Reflections*. Morgan Kaufmann, 2003.

[3] Allen H. Dutoit Bernd Bruegge. *Object-Oriented Software Engineering: Using UML, Patterns and Java,*. Prentice Hall, 2003.

[4] Michael Callaghan and Heiko Hirschmuller. 3-d visualisation of design patterns and java programs in computer science education. In *ITiCSE '98: Proceedings of the 6th annual conference on the teaching of*

computing and the 3rd annual conference on Integrating technology into computer science education, pages 37–40, New York, NY, USA, 1998. ACM Press.

[5] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman. *Readings in information visualization: using vision to think*. Morgan Kaufmann Publishers Inc., 1999.

[6] Siobhan Clarke and Elisa Baniassad. *Aspect-Oriented Analysis and Design*. Addison-Wesley Professional, 2005.

[7] Stephan Diehl. *Revised Lectures on Software Visualization, International Seminar*. Springer-Verlag, 2002.

[8] J. Dong and S. Yang. Visualizing design patterns with a uml profile. In *Proceedings of the IEEE Symposium on Human Centric Computing Languages and Environments*, pages 123–125, Auckland, New Zealand, 2003. IEEE Computer Society Press.

[9] Jing Dong, Sheng Yang, and Kang Zhang. Visdp: A web service for visualizing design patterns on demand. In *ITCC '05: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*, pages 385–391, Washington, DC, USA, 2005. IEEE Computer Society.

[10] Peter Eades and Kang Zhang. *Software Visualisation*, volume 7 of *Software Engineering and Knowledge Engineering*. World Scientific, 1996.

[11] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[12] Susan Hunston. *Corpora in Applied Linguistics*. Cambridge University Press, 2002.

[13] Ivar Jacobson and Pan-Wei Ng. *Aspect-Oriented Software Development with Use Cases (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2004.

[14] Dean F. Jerding. Visualizing patterns in the execution of object-oriented programs. In *CHI '96: Conference companion on Human factors*

in computing systems, pages 47–48, New York, NY, USA, 1996. ACM Press.

[15] Rilla Khaled, Anna Maria Luxton, James Noble, Leo Ferres, Judy Brown, and Robert Biddle. Visualisation for learning oop, using aop and eclipse. In *OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 178–179, New York, NY, USA, 2004. ACM Press.

[16] Rilla Khaled, James Noble, and Robert Biddle. Inspectj: program monitoring for visualisation using aspectj. In *ACSC '03: Proceedings of the twenty-sixth Australasian conference on Computer science*, pages 359–368, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[17] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[18] Ramnivas Laddad. *AspectJ in Action: Practical Aspect-Oriented Programming*. Manning Publications Co., Greenwich, CT, USA, 2003.

[19] Stuart Marshall, Kirk Jackson, Craig Anslow, and Robert Biddle. Aspects to visualising reusable components. In *Proceedings of the Australian Symposium on Information visualisation*, pages 81–88. Australian Computer Society, Inc., 2003.

[20] Stuart Marshall, Kirk Jackson, Robert Biddle, Michael McGavin, Ewan Tempero, and Matthew Duignan. Visualising reusable software over the web. In *Proceedings of the Australian Symposium on Information visualisation*, pages 103–111. Australian Computer Society, Inc., 2001.

[21] James Noble and Robert Biddle. Visualising 1,051 visual programs module choice and layout in the nord modular patch language. In *APVis '01: Proceedings of the 2001 Asia-Pacific symposium on Information visualisation*, pages 121–127, Darlinghurst, Australia, Australia, 2001. Australian Computer Society, Inc.

[22] James Noble and Robert Biddle. Program visualisation for visual programs. In *AUIC '02: Proceedings of the Third Australasian conference on User interfaces*, pages 29–38, Darlinghurst, Australia, Australia, 2002. Australian Computer Society, Inc.

[23] Alex Potanin, James Noble, Marcus Frean, and Robert Biddle. Scale-free geometry in oo programs. *Commun. ACM*, 48(5):99–103, 2005.

[24] Roger S Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 2004.

[25] John T. Stasko, Marc H. Brown, and Blaine A. Price. *Software Visualization*. MIT Press, 1997.

[26] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 1998.

[27] John Vlissides. *Pattern Hatching: Design Patterns Applied*. Addison-Wesley, 1998.

[28] Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann, 2000.

[29] Kang Zhang. *Software Visualization: From Theory to Practice*. Kluwer Academic Publishers, 2003.