Software Evolution and 'Light' Semantics

Extended Abstract

Dewayne E. Perry
Bell Laboratories
600 Mountain Ave
Murray Hill, NJ 07974 USA
+1.908.582.2529
dep@research.bell-labs.com

Keywords
Inscape Environment, Software Complexity, Software Evolution, Implications of Changes, Light Semantics, Interface Specifications, Software Composition

1 INTRODUCTION

The motivation for Inscape [4] came from my experience as a programmer, designer and architect. There were two major (and inter-related) problems that I encountered while building software systems where I had to use components built by other people: the pieces often did not fit when I put them together and changing code often produced surprising and unexpected results.

The first problem was due primarily to the informality and often incompleteness of component interfaces. The second problem was due ultimately to the complexity of the software and an inability to foresee or determine the consequences of changes. These problems result from three essential [1] and intertwined properties of building software systems: composition, evolution [2] [7] and complexity [5].

In coming to grips with the problem of composition, using formal interface specifications is the obvious choice. Enhancing the syntactic interfaces with semantic information is one useful way of expressing the intent of the interface provider and enabling the user to have all the information necessary to its correct and effective use.

How to attack the problem of evolution is not as obvious. The approach I took in the Inscape experiment was to use the specifications *constructively* in order to determine and maintain semantic dependencies [3]. The metaphor is that of a hardware chip: the dependencies are the pins that are used. Keeping track semantically as to how the interfaces are used is the analog of expressing the interface creator's intent: it is capturing the users intent. Given that both interfaces and implementations evolve, keeping track of the dependencies enables the environment to help in understanding the effects of changes and where those effects take place.

ICSE '99 Los Angeles CA

The primary question then is how to make it work. An important clue to that question comes from considering the problem of complexity.

2 COMPLEXITY & 'LIGHT' SEMANTICS

In [5] I delineated two kinds of complexity. One kind of complexity is that of the intricacies of some algorithms and data structures. Here the complexity is analogous to a Bach four-voice fugue where there are very intricate and intertwining multi-dimensional constraints (in the case of a fugue, melodic and harmonic, or horizontal and vertical, constraints). One cannot make arbitray changes without having (usually) disastrous consequences. All of the constraints must be understood, and in this case, that means gaining *deep* insight and understanding before successful changes can be made. Reasoning about this type of complexity is just plain hard. It is often hard for even the creator of the software to remember all the lines of reasoning that resulted in the final state.

However, there is another and completely different type of complexity that has different characteristics. This is the complexity that arises from the sheer wealth, or mass, of details. Here comprehension is hindered by the problem of scale, not the inherent complexity of the individual details. The analogy here is more to a Strauss tone poem or a Mahler symphony where the complexity is due to the mass of detail rather than a fugue's intricate complexity (though there may be that kind of complexity underlying or buried in the mass of details).

My claim is that it is this latter kind of complexity that dominates most of the systems we build. There are parts of the systems that have the intricate complexity, but the rest of the system has the wealth of detail complexity. In support of this thesis, I offer the evidence of an error study of a release in the evolution of a large, realtime software system [8]. Overwhelmingly the faults

Proceedings of the 21st International Conference on Software Engineering (ICSE'99)
0270-5257/99 $ 10.00 © 1999 ACM

are the normal development faults with most of them fairly shallow and easy to find and solve. The hard ones were predictable: race conditions and performance.

This wealth of small details as the dominant form of complexity suggests that one needs to consider managing the small details rather than automating deep insight. For these small details, a much *lighter* form of semantic approach should be possible than full automated theorem proving but which goes beyond the current available forms of type checking.

There are various forms this idea of *light* semantics might take. One form might be using only part of the semantic information or use it in rather simple but automated ways. Another form might be approximations to aid the developer (often interactively and iteratively) in developing an understanding of the code involved.

In Inscape, I chose the former. The underlying thesis of the semantic interconnections and the semantic propagation [6] is that it is made up of a large number of very small theorems rather than a very small number of large theorems. Supporting this is the fact that there is no notion of an invariant in the underlying logic of Inscape — just the small details of what is needed before and what is guaranteed after a operation has been executed.

Thus it is this idea of light semantics that was used in Inscape as the basis for the constructive use of interface specifications and semantic dependency based evolution.

3 COMPOSITION & EVOLUTION

One of the primary contributions of Inscape to interface research was the introduction of obligations and multiple results. Obligations are an often unmentioned side effect of an operation. They may arise for a variety of reasons, but typically are not made explicit. Multiple results are needed for any kind of software where exceptions are needed (which is most of it). Adding these two elements to interfaces then provided a solid base for describing interfaces for use in production (rather than academic or toy) software.

There is only one basic rule in constructing programs: all preconditions and obligations must be satisfied within an implementation or propagated to the interface. There are, however, constraints on what may be propagated to the interface. These constraints result in what I called precondition ceilings and obligation floors. Those preconditions and obligations which could not be propagated to the interface represented semantic problems that needed to be resolved.

By keeping track in the construction of an implementation where (and how) the preconditions and obligations were satisfied, changes to either the interfaces used or to the implementations could enable the environment to determine how those changes affected an encompassing interface and thus enable the programmer to understand the effects of those changes.

4 RELATED WORK

There have been a number of projects that I know of that have been based on Inscape. Among these was an internally sponsored research project at Anderson Consulting called Software Interface Specification and Analysis. This project evolved into a Component-Based Software Engineering (CBSE) project, co-sponsored by the U.S. government Department of Commerce and Andersen Consulting. This work in turn was incorporated in their internal component framework and deployed in many of their client companies. This project in turn influenced David Curtis, now one of the authors of the Corba Component Model. See the papers in Appendix B by Jim Ning and/or Wojtek Kozaczynski.

Daniel Jackson's PhD thesis and some of his subsequent work was influenced by my 'light' semantic approach in Inscape. Aspect uses partial specifications provided in an annotation language which can be efficiently checked. The Aspect checker then uses a dependency analysis to check the code against the annotations. His more recent work also fruitfully explores the space of light semantics to provide various kinds of analyses and models.

Don Batory's work takes an approach very similar to my propagation logic in generating systems from components. These components are annotated with various properties which are then used compositionally to stitch the components together in a consistently well-formed composition.

Appendix B has further references for work either inspired by Inscape or related work exploring this exciting space of light semantics.

5 SUMMARY

The space of light semantics is an exceedingly fruitful field for research. Inscape was one of the first projects to explore this space and it has been followed some very interesting work in the subsequent years. The space is by no means exhausted — indeed it has been not been explored much at all. It is a ripe area for research that has too long been overlooked. And for me, the spirit of Inscape continues in my software architecture work.

The viewgraphs for this talk as well as all the relevant Inscape papers (See Appendix A below) can be found at my web site: www.bell-labs.com/user/dep/

ACKNOWLEDGEMENTS

First, I thank Jim Christ, Bill Schell, Steve Popovich, Peggy Quinn and Helen Diamontitus for their various contributions to Inscape. And of course, special thanks go to Prof. Gail Kaiser for our long and fruitful collaboration on Infuse, the change management part of

Proceedings of the 21st International Conference on Software Engineering (ICSE'99)
0270-5257/99 $ 10.00 © 1999 ACM

Inscape.

Next, I thank Don Batory, Prem Devanbu, Daniel Jackson, Wojtek Kozaczynski, Jim Q. Ning, and David Notkin for their various clarifying discussions and contributions to this talk and abstract.

Finally, I thank the ICSE99 Program Committee for selecting the Inscape paper as the most influential paper from ICSE11.

REFERENCES

[1] Frederick P. Brooks, Jr. No Silver Bullet: Essence and Accidents of Software Engineering. *Computer*, 20(4):10-20, (April 1987),

[2] M. M. Lehman and L. A. Belady. *Program Evolution. Processes of Software Change.* APIC Studies in Data Processing No. 27. London: Academic Press, 1985.

[3] Dewayne E. Perry. Software Interconnection Models. Proceedings of the 9th International Conference on Software Engineering, Monterey, CA, March 1987.

[4] Dewayne E. Perry. The Inscape Environment. *The Proceedings of the Eleventh International Conference on Software Engineering*, May 1989, Pittsburgh, PA.

[5] Dewayne E. Perry. Industrial Strength Software Development Environments. *Proceedings of IFIP '89 - 11th World Computer Congress*, August 1989, San Francisco, CA.

[6] Dewayne E. Perry. The Logic of Propagation in The Inscape Environment. Proceedings of SIGSOFT '89: Testing, Analysis and Verification Symposium, Key West FL, December 1989.

[7] Dewayne E. Perry. Dimensions of Software Evolution (Invited Keynote Paper). *International Conference on Software Maintenance 1994*, Victoria BC, September 1994.

[8] Dewayne E. Perry and Carol S.Steig. Software Faults in Evolving a Large, Real-Time System: a Case Study. 4th European Software Engineering Conference – ESEC93, Garmisch, Germany, September 1993.

APPENDIX A: Inscape Bibliography

Dewayne E. Perry and W. Michael Evangelist. "An Empirical Study of Software Interface Errors", Proceedings of the International Symposium on New Directions in Computing, IEEE Computer Society, August 1985, Trondheim, Norway, pages 32-38.

Dewayne E. Perry. "Position Paper: The Constructive Use of Module Interface Specifications", Third International Workshop on Software Specification and Design. IEEE Computer Society, August 26-27, 1985, London, England.

Dewayne E. Perry. "Tools for Evolving Software", Proceedings of the 2nd International Workshop on The Software Process and Software Environments, March 1985, Cota De Casa, Trabuco Canyon, CA, Software Engineering Notes 11(4):134-135 (August 1986).

Dewayne E. Perry. "The Construction of Robust, Fault-Tolerant Software in the Inscape Environment", AT&T Fault-Tolerance Symposium, September 1986.

Dewayne E. Perry. "The Iteration Mechanism in the Inscape Environment", Proceedings of the 3rd International Software Process Workshop: Iteration in the Software Process, November 1986, Breckenridge CO, pages 49-52.

Dewayne E. Perry. "Programmer Productivity in the Inscape Environment", The Proceedings of GLOBECOM '86, December 1986, Houston TX, pages 0428-0434 (12.6.1-12.6.7).

Dewayne E. Perry and W. Michael Evangelist. "An Empirical Study of Software Interface Faults — An Update", Proceedings of the Twentieth Annual Hawaii International Conference on Systems Sciences, January 1987, Volume II, pages 113-126.

Dewayne E. Perry and Gail E. Kaiser. "Infuse: A Tool for Automatically Managing and Coordinating Source Changes in Large Systems", Proceedings of the 1987 ACM Computer Science Conference, February 17-19, 1987, St. Louis MO.

Dewayne E. Perry. "Software Interconnection Models", Proceedings of the 9th International Conference on Software Engineering, Monterey, CA, March/April 1987. pp 61-69. *Best Paper, ICSE9.*

Dewayne E. Perry. "Version Control in the Inscape Environment", Proceedings of the 9th International Conference on Software Engineering, March/April 1987, Monterey CA.

Gail E. Kaiser and Dewayne E. Perry. "Workspaces and Experimental Databases: Automated Support for Software Maintenance and Evolution", Conference on Software Maintenance-1987, Austin, TX, September 1987. pp 108-114.

Dewayne E. Perry and Gail E. Kaiser. "Models of Software Development Environments". The Proceedings of the Tenth International Conference on Software Engineering, April 1988, Raffles City, Singapore. IEEE Transactions on Software Engineering, 17:3 (March 1991).

Dewayne E. Perry, James T. Krist, and William W. Schell. "The Inscape Environment and the Design of Finite State Machines in SDL". 5ESS Software Development Environment Conference, Naperville IL, November 1988.

Helen Diamontitus and Dewayne E. Perry. "Economic Modeling of the Inscape Environment". April 1989.

Dewayne E. Perry. "The Inscape Environment". The Proceedings of the Eleventh International Conference on Software Engineering, May 1989, Pittsburgh, PA.

Dewayne E. Perry. "Industrial Strength Software Development Environments". Proceedings of IFIP '89 - 11th World Computer Congress, August 1989, San Francisco, CA. *Invited Keynote.*

Proceedings of the 21st International Conference on Software Engineering (ICSE'99)
0270-5257/99 $ 10.00 © 1999 ACM

Gail E. Kaiser, Dewayne E. Perry and William M. Schell. "Infuse: Fusing Integration Test Management with Change Management" Proceedings of COMSAC 89, Kissimmee FL, September 1989

Dewayne E. Perry. "The Logic of Propagation in The Inscape Environment", Proceedings of SIGSOFT '89: Testing, Analysis and Verification Symposium, Key West FL, December 1989.

Dewayne E. Perry and Steven S. Popovich. "Inquire: Predicate-Based Use and Reuse". Specification Driven Tools Conference, AT&T Bell Laboratories, October 1989.

Dewayne E. Perry and Gail E. Kaiser. "Adequate Testing and Object-Oriented Programming". Journal of Object-Oriented Programming, January-February 1990.

Dewayne E. Perry and Gail E. Kaiser, "Making Progress in Cooperative Transaction Models", IEEE Bulletin on Data Engineering, 14:1 (March 1991).

Stephen S. Popovich, William M. Schell, and Dewayne E. Perry. "Experiences with an Environment Generation System", Proceedings of the 13th International Conference on Software Engineering, May 1991, Austin TX.

Dewayne E. Perry. "Dimensions of Consistency in Source Versions and System Compositions — A Position Paper" Proceedings of the 3rd Workshop on Software Configuration Management, Trondheim, Norway, June 1991.

Dewayne E. Perry and Steven S. Popovich, "Inquire: Predicate Based Use and Reuse", Knowledge-Based Software Engineering Conference, Chicago IL, September 1993.

Dewayne E. Perry and Carol S.Steig, "Software Faults in Evolving a Large, Real-Time System: a Case Study", 4th European Software Engineering Conference - ESEC93, Garmisch Germany, September 1993. *Invited Keynote.*

Dewayne E. Perry, "System Compositions and Shared Dependencies", 6th Workshop on Software Configuration Management, ICSE18, Berlin Germany, March 1996.

APPENDIX B: Some Related Work

This should by no means be considered an exhaustive bibliography. Instead it represents some of the work that is either directly or indirectly influenced by Inscape or, like Inscape, explores this space of 'light' semantics.

D. Batory, G. Chen, E. Robertson, and T.Wang, "Design Wizards and Visual Programming Environments for Generators" Int. Conference on Software Reuse, June 1998 (Victoria, Canada).

Don Batory and Bart J. Geraci. "Composition Validation and Subjectivity in GenVoca Generators". IEEE Transactions on Software Engineering, February 1997, 67-82.

Don Batory, Vivek Singhal, Jeff Thomas, Sankar Dasari, Bart Geraci, and Marty Sirkin. "The GenVoca Model of Software-System Generators". IEEE Software, September 1994.

Don Batory and Sean O'Malley. "The Design and Implementation of Hierarchical Software Systems with Reusable Components". ACM Transactions on Software Engineering and Methodology, 1(4):355-398, October 1992.

Alexander Borgida and Premkumar Devanbu. Component inter-operability—putting "DL" to "IDL" International Conference on Software Enginering - 1999, May 1999.

Francois Bronsard, Douglas Bryan, Wojtek Kozaczynski, Edy S. Liongosari, Jim Q. Ning, Asgeir Olafsson, and John Wetterstrand "Toward Software Plug-and-Play", Proceedings of the Symposium on Software Reusability, Boston, Massachusetts, May 18-23, 1997.

Robert Callahan and Daniel Jackson. "Lackwit: A Program Understanding Tool Based on Type Inference". Proc. International Conference on Software Engineering, Boston, MA, May 1997.

Wojtek Kozaczynski and Jim Q. Ning. "Concern-Driven Design for a Specification Language Supporting Component-Based Software Engineering". Proceedings of the Eighth International Workshop on Software Specification and Design, Schloss Velen, Germany, March 22-23, 1996.

Daniel Jackson. "An Intermediate Design Language and its Analysis". Proc. ACM Conference on Foundations of Software Engineering, Florida, November 1998.

Daniel Jackson. "Aspect: Detecting Bugs with Abstract Dependences". ACM Transactions on Software Engineering and Methodology, 4(2):109-145, (April 1995).

Daniel Jackson and Allison Waingold "Lightweight Extraction of Object Models from Bytecode". Proc. International Conference on Software Engineering, Los Angeles, CA, May 1999.

Daniel Jackson and Craig A. Damon. "Elements of Style: Analyzing a Software Design Feature with a Counterexample Detector". IEEE Transactions on Software Engineering, 22(7):484-495 (July 1996).

G.C. Murphy, D. Notkin, K. Sullivan. "Software reflexion models: Bridging the gap between source and high-level models". Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, WashingtonDC, October 1995.

Jim Q. Ning. "Component-Based Software Engineering". Proceedings of the Fifth International Symposium on Assessment of Software Tools, Pittsburgh, Pennsylvania, June 2-5, 1997.

Jim Q. Ning "A Component-Based Software Development Model". Proceedings of the Twentieth Annual International Computer Software and Applications Conference, Seoul, Korea, August 19-23, 1996.

Jim Q. Ning, Kanth Miriyala, and Wojtek Kozaczynski. "An Architecture-driven, Business-specific, and Component-based Approach to Software Engineering". Proceedings of the International Conference on Software Reusability, Rio de Janeiro, Brazil, Nov. 1-4, 1994.

Proceedings of the 21st International Conference on Software Engineering (ICSE'99)
0270-5257/99 $ 10.00 © 1999 ACM