Using Transformation Systems for Software Maintenance and Reengineering

Ira D. Baxter
Semantic Designs, Inc.
idbaxter@semdesigns.com

Abstract

Software maintenance costs dominate software engineering costs, partly because most such engineering is done manually. Program Transformation tools leverage an engineer-provided base of "transforms" (a kind of generative reuse of programming knowledge), to automate analysis, modification, and generation of software, enhancing productivity and quality over conventional methods. This tutorial provides a complete overview of Program Transformation, from theory to implementation to application. Several real transformation systems will be examined, with application examples including automated detection and removal of duplicate code from large systems, and the potential for semi-automated refactoring of large object frameworks. The tutorial progresses from introductory to intermediate, but all the necessary background will be provided, so attendees need only basic software engineering knowledge and motivating experience modifying software.

1 Transformation Systems

Transformation systems are software tools that “rewrite” constellations of concepts (characters, strings, trees, graphs) into alternative constellations. (The basic technology idea goes back to Emil Post’s 1943 model of computation as rewriting strings, but the foundations are found in mathematics). This rewriting capability has long considered [11] as a key enabler supporting

- Analysis of software
- Modification of software
- Generation of software

Rewriting provides the basis for automation. Applying previously coded rewriting rules provides significant additional leverage by reusing hard-won problem concepts, analysis and implementation methods.

Transformation system technology has matured to the point where these activities are practical on large scale, production software systems, and offer large productivity and quality increments to engineering organizations using them [1]. Practical tools are now beginning to appear in the marketplace. *We predict these tools will cause a paradigm shift for software engineering from ad hoc design and manual coding of programs and tests, to specification of problems and solutions, and encoding implementation knowledge. The ability to make massive, reliable changes to software will fundamentally change the way organizations design and deliver software systems*

This new class of tools will become a part of every software engineer's toolkit over the next decade, just like editors, compilers, linkers and debuggers. The tutorial will provide the researcher and the practitioner alike with the necessary background to understand how these tools work, how to determine the capabilities and limitations of a particular system, and how to compare and therefore rationally choose among different transformation systems.

2 Technology Foundations

Practical transformation systems are extremely generalized compilers. They must:

- Parse a source language or specification
- Build a computer-internal-representation
- Carry out analyses for well-formedness, and/or desired/undesired properties of the spec based on local information and information flows
- Carry out modifications to the representation according to previous analyses, to either
  - Enhance the program (speed, space, quality)
  - Translate to another language
  - Abstract from low level concepts
- Regenerate program source text from updated internal models.

In addition, for use in practical software engineering, such tools must scale well in many axes:

- Size of the program specification (thousands of files, millions of lines)
- Number of and variety of languages they can process, both in general and in the same session
- Computational horsepower to enable analysis and modification of large systems
- Ability to interact with multiple engineers over long periods of time
- Sophistication of analysis/change methods
- Ability to acquire new knowledge (rewrites, analysis and modification methods)[4]

The tutorial will cover these technology foundations, both in concept and in mechanism, and show how such components play together to achieve the overall effects. A conceptual vocabulary for clearly identifying transformation system components will be provided and act as basis of comparison of a number of practical systems. The author's scalable DMS Reengineering Toolkit will be used to illustrate a number of the foundations (sophisticated parsing/prettyprinting, multiple domain languages [10], parallel computational support. etc), and how they are integrated.

3. Applications of Transformation Systems

The value of transformation systems lies in their agility to serve an amazing variety of software engineering purposes. The tutorial will generally show how these tools can be harnessed for:

- Reverse Engineering (structure and concept extraction)
- Arbitrary Analyses
- Reengineering (reshaping/reorganizing code)[8]
- Document extraction
- Software Porting (new languages/platforms) [13]
- System Optimization (space, parallelization,…)[7]
- Code generation from specifications
- Field size overflow repair (Y2K, SSNs, …)
- Applying Design Patterns automatically [12]

The tutorial will examine several applications of special interest in detail:

- Code generation from Finite State Specifications and/or XML DTDs
- Clone Detection and Removal, which finds and removes typically 10% of the source code volume from *any* system of modest scale[3]
- Refactoring, (reengineering object-oriented systems), and how automated tools are required to realize the promises of the refactoring/XP [5] community

Finally, we will examine a theory of software change[2] specification and automated insertion/management being implement with DMS.

4. The Presenter

Ira Baxter has been building system software since 1969. He acquired his Ph.D. with emphasis on software engineering and reuse from the University of California at Irvine in 1990. He has worked with a number transformation systems starting with Draco [10] in 1975, and is presently the architect of the DMS, and designer of the PARLANSE parallel programming language in which DMS is implemented.

Dr. Baxter has been invited speaker at SSR'99, coChair of the 1997 International Conference on Software Reuse, and Program coChair of the Working Conference on Reverse Engineering, and has been a PC member of the International Conference on Software Maintenance for a number of years.

References

[1] http://www.program-transformation.org

[2] I. Baxter and C. Pidgeon, "Software Change Through Design Maintenance",. International Conference on Software Maintenance, IEEE Press, 1997.

[3] I. Baxter, et. al, "Clone Detection Using Abstract Syntax Trees", International Conference on Software Maintenance, IEEE Press, 1998.

[4] M.G.J. van den Brand, M.P.A. Sellink, and C. Verhoef, "Generation of components for software renovation factories from context-free grammars", *Science of Computer Programming*, 36(2-3):209-266, 2000

[5] M. Fowler, Refactoring, Addison-Wesley March 2000

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995

[7] N Jones, "An introduction to partial evaluation", *ACM Computing Surveys*, 28(3):480-503, September 1996.

[8] L. Markosian, P. Newcomb, R. Brand, S. Burson, T. and Kitzmiller, "Using and Enabling Technology to Reengineer Legacy Systems", *Communications of the ACM*, 37 (5) pp. 58--71. 1994

[10] J. Neighbors, "The Draco Approach to Constructing Software from Components", IEEE Transactions on Software Engineering 10(5):564-574, 1984.

[11] H. Partsch, *Specification and Transformation of Programs*, Springer-Verlag, 1990

[12] L. Tokuda, D. Batory, "Automated Software Evolution via Design Pattern Transformations", TR-95-06 CS Dept., University of Texas at Austin, 1995

[13] R. C. Waters. "Program translation via abstraction and reimplementation", *IEEE Transactions on Software Engineering*, 14(8):1207-1228, August 1988.