Perpetual Testing Research at Purdue
Quarterly Technical Report Vol 2, Number 2

Michal Young
University of Oregon
1202 Department of Computer Science
Eugene, Oregon 97403-1202
michal@cs.uoregon.edu

For the period: 31 April 1998 through 30 July 1998

Cooperative agreement: F30602-97-2-0034

Prepared for: U.S. Air Force, Air Force Materiel Command, Rome Laboratory

Abstract

The Perpetual Testing project goal is to develop technologies to support seamless, perpetual analysis and testing of software through deployment and evolution. Whereas the current dominant paradigm treats testing as a phase that succeeds development and precedes delivery, the perpetual testing projects is building a foundation for treating analysis and testing as on-going activities to improve quality assurance without pause through several generations of product, in the development environment as well as the deployed environment. Improvements to existing technologies focus largely on scalability and incrementality for large evolving systems, attaining and maintaining adequate adherence of all software artifacts to relations captured by a rich web of hypercode links, including dependence relations among software components and among properties and analysis techniques.

The perpetual testing project is a collaboration of researchers at the University of Massachusetts, the University of California at Irvine, and University of Oregon. Perpetual testing is part of the High Assurance cluster of DARPA-sponsored EDCS research.

1 Key Project Events During Quarter

TABLE 1. Key project meetings

Title	Site	Dates	Attendees
EDCS “demo days” and PI workshop	Baltimore, Maryland	July 20-24	Young, Cheng

The main event of this period was the EDCS “Demo Days” workshop in Baltimore, and as in previous years much of our effort was focused on preparing for it. We demonstrated a combination of newly developed tools (particularly a new architecture refactoring tool to support two-phase testing) and some tools that had been shown at the previous Demo Days workshop.

Other presentations. Michal Young served as a panelist on the topic of “software as a profession” at the 1998 Motorola software engineering symposium (a company-wide event for Motorola software developers) near Chicago. Michal Young gave a short presentation on two-phase testing at the NSF/CNR workshop on the Role of Software Architecture in Analysis and Testing (Rosatea) in Marsala, Italy.

2 Results

Two-phase testing. Two-phase testing refers to a combination of static analysis of a logical design model coupled with dynamic testing to demonstrate code conformance. It differs from directly testing an implementation for desired properties (which is being pursued in other sub-projects of Perpetual Testing) in the “indirect” approach of statically verifying a model and then using the model to generate test oracles, thus the term “two-phase.” Direct and two-phase testing are not exclusive options, but are suitable for different properties. Two-phase testing is aimed particularly at properties that are dependent on non-deterministic and timing-dependent execution, such as race conditions and potential deadlock in concurrent software.

In the current period, Yung-Pin Cheng finished implementation of the revised version of an architectural refactoring tool, preparing it for EDCS demo days. In addition, he refined his approach to refactoring systems for inductive verification, showing that in some cases refactoring can make possible inductive verification where it is otherwise impossible to find an invariant process on which to base the induction.

Graph interchange and attribution Many objects manipulated by software development tools can easily be represented as directed, attributed graphs (DAGs[1]). However, even when two tools manipulate objects that are representable as graphs, it is common that they cannot easily interoperate by exchanging graph objects. ACME is suitable for exchanging certain kinds of graph objects which rep-

1. The acronym “dag” is commonly used for directed, *acyclic* graphs, but here the “A” stands for “attributed” and not for “acyclic.”

resent system architecture, as the fc2 format is suitable for exchanging process graphs, but some kinds of generic graph manipulation are independent of whether the graph represents an architecture, program control flow, synchronization structure, or something else entirely. Our work on graph interchange and manipulation addresses this generic level which can be thought of as “one level down” from domain-specific graph representations such as ACME. We have adopted the “DOT” textual notation for DAGs, originally devised at AT&T Bell Labs and widely used for graph visualization. In addition to several small translator tools (e.g., from ACME to DOT and back), our main effort is to provide a generic, programmable system for computing computing and attributing graphs using flow analysis.This project is described at *http://www.cs.uoregon.edu/research/perpetual/GenSet.html*.

Graduate research assistant Craig Kaes finished implementation of the first version of GenSet, a generic graph attribution tool in early June. This tool supports programmable computation of graph attributes from fixed point equations; we view it as roughly analogous to “awk” for attributed graphs rather than simple text streams. The underlying algorithms are the same as flow analysis computations used in compilers, but their application can be completely different. In particular, we plan to reimplement our previous structure recovery tools using this new generic attribute computation mechanism.

3 Conclusions

Our difficulties in establishing a subcontract relationship with Purdue were finally resolved in this period, unfortunately too late to staff up in preparation for Demo Days. As a result our demonstrations were less polished than they might have been, but fortunately we were able to demonstrate some new capabilities in two-phase testing as well as repeating some demonstrations from the previous year. Perhaps the biggest achievement of this period was completion of the first prototype of GenSet, even though it was completed very late in the period and did not make it into our “official” demonstration in Baltimore (we did demonstrate it informally to several attendees).