# Perpetual Testing Research at Purdue
# Quarterly Technical Report Vol 2, Number 1

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

For the period: 31 Janury 1998 through 30 April 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 PI “dry run” meeting | Los Angelas | March 16-19 | Young |

The main project meeting of the period was the EDCS Architecture and High-Assurance cross-cluster working group meeting in Austin.

**Other presentations.** Michal Young served as program chair of the ACM International Symposium on Software Testing and Analysis in St. Petersburg, Florida, March 2-4. Young also presented a tutorial on software testing and analysis at the International Conference on Software Engineering in Kyoto, Japan, on April 20.

## 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 continued reimplementation of his architecture refactoring tool according to the design arrived at in the immediately prior period. ACME import/export capabilities were improved, and an interface to external finite-state verification tools using the standard fc2 format for process graph interchange was completed. The graphical user interface for the revised tool (using Java) is under development.

**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 represent 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”

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

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*.

Through the current period, Craig Kaes contiunued design and implementation of a programmable flow analysis engine for generic DAGs in the DOT format. This tool will support 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

We remained understaffed through this period due to continued delays in establishing a subcontract with Purdue, but continued to make good progress in two parts of the project, two-phase testing and graph interchange, partly by “borrowing” funds at Oregon to pay one of the graduate assistants and cover travel expenses until they can be properly charged back to the Perpetual Testing project.