Perpetual Testing Research at Purdue
Quarterly Technical Report Vol 1, Number 4

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

For the period: 31 October 1997 through 30 January 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 High Assurance and Architecture cross-cluster meeting	Austin, Texas	November 10-12	Young

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

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 began reimplementation of his architecture refactoring tool according to the design arrived at in the immediately prior period. Initial ACME import/export capabilities was implemented. An interface to external finite-state verification tools using the standard fc2 format for process graph interchange was begun. Additionally, an approach to refactoring architectures for inductive verification has been worked out and applied, using existing tools, to a few small examples; implementation of tool support and more significant case studies will begin shortly.

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" from domain-specific graph representations such as ACME.

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

In the Unix world, streams of minimally structured text is an exchange format for interoperation of many simple tools, and this has in particular made possible or encouraged the development of simple "tool fragments" that can be flexibly combined to perform non-trivial tasks. Some of these tool fragments, like "expand", perform simple, relatively fixed transformations, while others like "awk" provide a programmable platform for producing application-specific processing. The goal of an exchange format for DAGs is similar: to encourage packaging of useful functionality as "tool fragments" that can be easily and flexibly combined, and to make practical provision of powerful mechanisms that can be customized with many application-specific policies.

While we have adopted domain-specific representations such as ACME and fc2 where appropriate, for generic graph interchange and manipulation we have chosen the "DOT" format developed by AT&T. The reasons for choosing DOT over other candidate representations are pragmatic: It is already supported by an excellent graph layout tools, it is widely available, and it is already used by many research and a few industrial projects.

In January, graduate research assistant Craig Kaes began design 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.

Structure Recovery As mentioned above, our prior work on recovering architectural structures from software will be subsumed by the current work on generic graph attribution. This will make it far more flexible, as well as serving as an example of the utility of generic tools for manipulating a common graph representation.

3 Conclusions

Although we were understaffed in this period due to delays in establishing a subcontract with Purdue, progress in the two-phase testing part of the project continues, and the previous work on structure recovery has been revived in a more general form.