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

Michal Young
Department of Computer Sciences
Purdue University[1]
West Lafayette, Indiana 47907-1398
michal@cs.uoregon.edu

For the period: 20 April 1997 through 31 July 1997

Cooperative agreement: F30602-97-2-0034

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

1. The principal investigator is currently at University of Oregon, but this report is for the period during which he was at Purdue University.

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 Purdue University. 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
2nd EDCS Program Workshop and Demo Days	Seattle, Washington	July 21-24	Young, Cheng
Arcadia research planning	Boulder, Colorado	June 30–July 2	Young

The key event of the quarter was the 2nd EDCS Demo Days workshop in Seattle, and much of the activity of the period was focused on preparing for the demonstration. Michal Young and Yung-Pin Cheng attended the meeting, where they demonstrated prototype tools for two-phase testing, residual testing, and view editing.

Other presentations. Michal Young gave an invited presentation on combining static analysis with testing at Honeywell Technology Center in Minneapolis, Minnesota on 6 May 1997. This presentation was made in conjunction with technical meetings with Steven Vestal and Jon Ward of Honeywell, to explore possibilities of joint research in which perpetual testing technologies would be incorporated in Meta-H.

Michal Young presented a technical paper at the 1997 International Conference on Software Engineering in Boston, May 19–23, 1997. The paper reported on progress in developing multi-formalism analysis tools.

Michal Young presented a report on perpetual testing research to industrial sponsors of the Software Engineering Research Center (SERC) at the “SERC Showcase” meeting, May 13–14, at Purdue. Attendees included representatives of Northrup-Grumman (Joint-Stars).

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.

An approach conceptually similar to two-phase testing has been used for communication protocols, but with strong assumptions and restrictions that made the approach unsuitable for general software systems. In particular, protocol conformance testing assumes a very simple relation between the

design model (which is typically called a “specification” in protocol testing literature) and an implementation. Two-phase testing is aimed at systems in which there is a complex, many-to-many relation between elements of a logical design model (which may be an instance of a software architecture) and an implementation. We have used the term “view editing” to describe management and manipulation of potentially complex relations between design and implementation artifacts; these relations are an essential facet of the web of “hypercode” relations that will form the basis for perpetual testing. We have previously reported construction of an initial prototype implementation of a view editing capability that graphically displays and manipulates designs expressed in the Ada-based design language accepted by the CATS/Pal analysis tool, which was developed as part of the Arcadia project. During the current period, Yung-Pin Cheng improved the prototype, making the tool more robust and attractive for demonstration at the EDCS Demo Days workshop.

We have previously reported an initial experiment in which test oracles for the Chimera 1.1 hypermedia system were generated semi-automatically from verified CATS/Pal models. Yung-Pin Cheng used this case study to evaluate and refine the dynamic testing apparatus for two-phase testing. He demonstrated at the EDCS Demo Days workshop a graphical interface which dynamically shows a portion of a state-machine representation of the model during testing, highlighting any violation of conformance to the verified model when it occurs.

Structure recovery J While ideally the relation between architectural design and implementation would be maintained at every point in software development and evolution, in practice it is often necessary to “recover” the relation from legacy systems. The “fanasci” tool takes an architectural design description and an implementation structure (derived from source code), together with a partial mapping from implementation modules to architecture-level components, and “fills in” additional relations between implementation and architecture that are implied by architectural design constraints. We previously reported a fanasci prototype that built up “uses” or “layers” hierarchies. In summer 1997 a tool for extracting Ada task call graphs (call graphs restricted to calling paths between tasks) was constructed using the core fanasci analysis engine. This tool was constructed partly to demonstrate the generality of the constraint propagation approach used in fanasci, and partly because we found it useful, and partly to provide a simple and understandable demonstration. In addition, fanasci was reconfigured to act as a server in a client-server configuration with a graphical interface. A simple graphical display client was prototyped in Java, using the GEF tool-kit from UC Irvine.

Residual test coverage monitoring. ”Residual” testing is testing that takes place after deployment. Work to date at Purdue has focused on monitoring of dynamic test thoroughness, as indicated by structural test coverage criteria, in deployed software. Residual coverage monitoring should indicate whether any test obligations which were not fulfilled in the development environment correspond to execution behaviors that occur in actual use. This could occur, for example, if an execution path that has been presumed infeasible by developers is in fact feasible. The primary question addressed by the current work is whether such monitoring can have sufficiently low impact on execution characteristics, particularly efficiency and responsiveness, to be acceptable to users.

Christina Pavlopoulou developed an initial prototype tool that provides residual statement coverage monitoring for Java programs, showing that residual monitoring can have very small run-time cost, at least for simple statement coverage monitoring.) and to gather data to evaluate design alternatives for path-oriented residual coverage monitoring. The prototype tool instruments Java.class files, and displays coverage using an emacs interface. Initial measurements using Java applets distributed by Sun microsystems show that the performance impact of residual monitoring is very low, well under 1% of execution time, because instrumentation in the uncovered portion of the code is rarely if ever executed. This initial prototype will be used to evaluate strategies for providing path-oriented coverage monitoring, which will be considerably more difficult to achieve with very low overheads.

Multi-formalism analysis. Flexible tools for analyzing a variety of different modeling formalisms, and for analyzing mixes of formalisms, is relevant to EDCS for two reasons. First, the "domain specificity" of architectural design notations suggests that different systems, and different parts of large systems, will be best modeled with different formalisms. Our investigation (in collaboration with Mauro Pezzè of Politecnico di Milano) is limited to state-space analysis tools for concurrent systems. Progress in implementation of prototype multi-formalism analysis tools was described in the immediately prior quarterly report; during the current period that progress was presented at the International Conference on Software Engineering.

Adaptive Parsing. During summer 1997 we collaborated with Jens Palsberg, who participates in a dynamic languages project under EDCS sponsorship, to build an "adaptive parsing" tool. This was a short-term collaboration that consisted primarily of supporting an undergraduate research assistant, Kevin Tao, who implemented the prototype tool. The purpose was to explore adaptive parsing as an alternative means of constructing tools for extracting information useful in analysis and testing, and for producing instrumented versions of programs for testing. The adaptive parsing tool permits very small changes in an existing language parser to provide added functionality such as insertion of probes.

Technology Insertion Activities. We stepped up interchanges with Honeywell Technology Center (Steve Vestal and Jon Ward) and with Northrup (Greg Johnson) to investigate potential uses of Perpetual Testing technologies in related projects. These investigations are still at a preliminary stage, and have involved mostly "mutual education" on potential applications and approaches, e.g., studying design documents for the Honeywell Meta-H compiled executive. Northrup has obtained and installed a copy of the CATS/Pal tool, and begun using it to learn about the compositional analysis approach with tutorial support (by email, so far) from Yung-Pin Cheng of Purdue (now Oregon).

3 Conclusions

Activities during this quarter were centered on preparation for the EDCS Demo Days workshop in Seattle. Progress was primarily in the form of extending prototype tools and bringing them to a state suitable for demonstration, rather than initiating new sub-projects. This has given us a good opportu-

nity to take stock of results so far and plan for the next major round of development and demonstration.

A complicating factor in the project is a move of the principal investigator from Purdue to the University of Oregon in August 1997. Arrangements for continuation of the project at Oregon are underway and should soon be complete, but some disruption was inevitable. The largest impact came from delaying hiring of a professional support programmer, since it would have been impractical to bring such a person from Purdue to Oregon. If subcontracting arrangements proceed as planned, our best estimate now is that a support programmer could be in place by January 1998, which will provide enough lead time to have an impact on development in preparation for the summer 1998 Demo Days as well as ongoing efforts to apply Perpetual Testing technology in Honeywell and/or Northrup projects.