

# An Empirical Evaluation of Maintaining Evolving Component-based Software with the UML

Ye Wu, Jeff Offutt and Yuqin Ding  
Information and Software Engineering Department  
George Mason University  
Fairfax, VA 22030, USA  
(703) 993-1651  
{wuye, ofut, yding1} @ise.gmu.edu

## 1. Introduction

Unlike traditional systems, many components of component-based software are maintained by third-party providers and the maintenance is invisible to component users. Even though the modified components may still keep the same interfaces, internal changes may adversely affect systems that use the components. Regression testing of the existing systems that integrate with modified components must be done carefully despite the fact that the component users do not have access to the source.

In a previous paper, we presented a method to model changes to components and then derive regression tests based on those changes [5]. This fast abstract presents results from an empirical evaluation of that regression testing technique. The changes are modeled by using the Unified Modeling Language (UML). We first establish a UML-based infrastructure to model different types of changes. Then regression-testing strategies are developed to try to validate these changes. This experimental evaluation applies the change modeling technique to a small system. The changes were modeled by UML collaboration diagrams and state charts, tests were generated based on several criteria, faults were inserted into (a changed version of) the implementation, and the tests were evaluated by their ability to detect the faults.

## 2. Background

This fast abstract takes a generic and simple view of components as independent pieces of software. *Interfaces* represent access points to components through which a client component can request services declared in the interface and provided by another component. We define an *event* to be an invocation of one interface through another interface. A component may be a server or a client, and some components may act in both roles.

Collaboration diagrams and state charts can be used to describe key aspects of the internal structures of components. Collaboration diagrams represent interactions among different objects in a component and state charts characterize internal behaviors of objects in a component.

Tests derived from the UML diagrams are then run on the implementation. Our testing uses test criteria defined in Ammann and Offutt [1, 2]. The criteria are based on decisions, which are defined using boolean-valued predicates comprised of one or more boolean-valued clauses. The *Combinational Coverage Criterion* (CoC) requires each predicate to be evaluated with all possible truth assignments for the constituent clauses. The *Correlated Active Coverage Criterion* (CACC) tests individual clauses. CACC is equivalent to the “masking” form of Multiple Condition Decision Coverage (MCDC), which the FAA requires for safety-critical parts of aeronautics software [4]. The *Edge Coverage Criterion* (EC) requires each edge to be taken (each predicate is both true and false), and the *Path Coverage Criterion* (PC) criterion requires all paths.

Software maintenance activities are classified as corrective, perfective and adaptive maintenance [3]. We consider corrective maintenance to require modifications to individual component classes, but not the overall structure of the component. Perfective and adaptive maintenance may change the internal structures of the components, but not the interfaces. The modifications to the UML diagrams determine which regression testing strategy to apply [5].

## 3. Empirical Evaluation

We performed an empirical evaluation of the technique on a small system for dispensing coffee. `CoffeeMachine.java` has 12 classes that implements the state chart shown in Figure 1. Changes to the UML diagrams were reflected in the implementation.

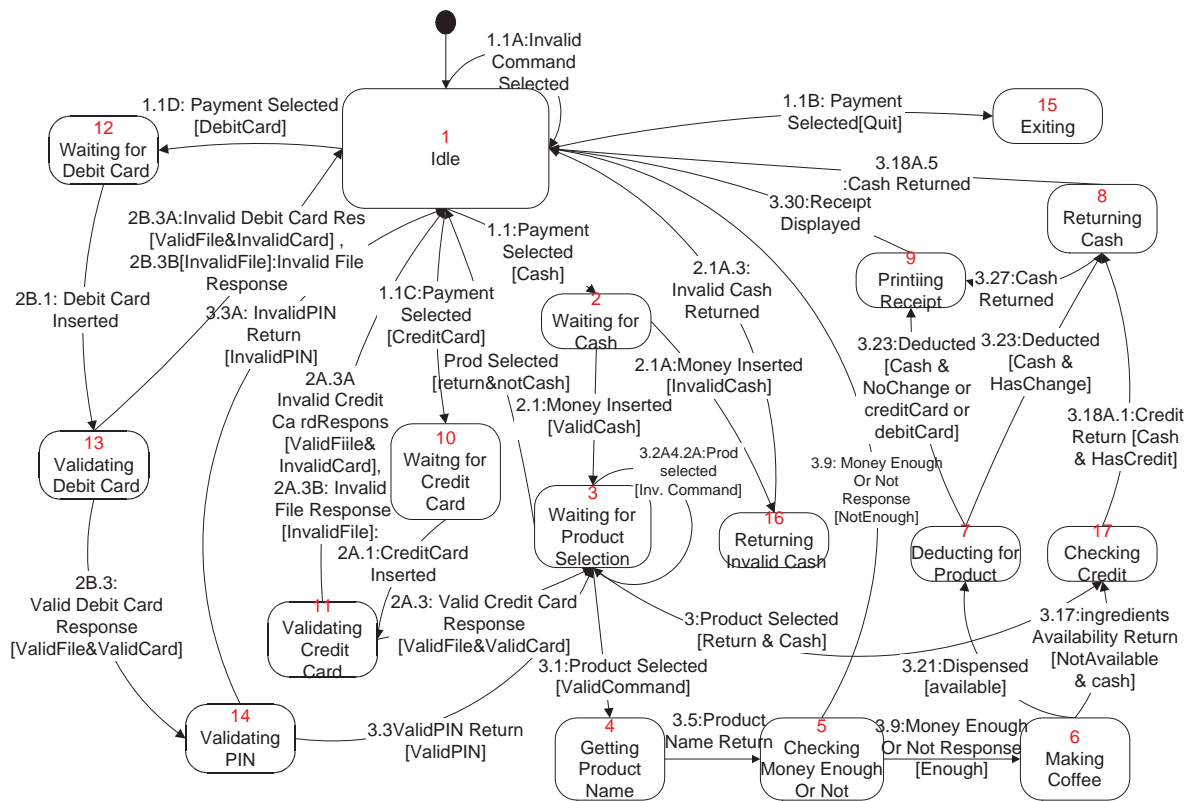


Figure 1. State chart of coffee dispensing machine system

Table 1. Faults found by each test set.

Source	Criteria	Tests	Faults	Faults Found	% Found
State chart	CoC	26	27	21	77.78%
State chart	CACC	17	27	21	77.78%
State chart	EC	7	27	15	55.56%
Collaboration	PC	20	27	19	70.37%
Collaboration	EC	7	27	13	48.15%

Existing tests were augmented with new tests. To avoid potential bias, the tasks were strictly divided with different persons creating the diagrams and implementation, generating faults, and generating tests.

The experiment assumed that the component providers carried out acceptable unit testing, a test set already exists from previous integration testing, and all modifications are correctly depicted on the UML diagrams.

#### 4. Results and Analysis

Table 1 shows that CoC, CACC, and EC were applied to state charts and PC and EC were applied to collaboration diagrams. In general, the tests from the state charts found more faults than those from the collaboration diagram.

The same six faults were **missed** by both CoC and CACC, and these faults were not found by any of the other test sets. Four of these six faults are related to specific im-

plementation decisions, so are unlikely to be found with design-based testing. The other two are more interesting. They are both the result of a variable that is left with an incorrect value after a transaction is completed. Thus, they could only be found by a test that includes two complete separate transactions carried out sequentially, a scenario that is not required by any of the logic-based test criteria. However, these faults could be found by a data flow criterion that caused the final def in the first transaction to be used in a second transaction.

#### References

- [1] P. Ammann and J. Offutt. *Coverage Criteria for Software Testing*. 2003. In preparation.
- [2] P. Ammann, J. Offutt, and H. Huang. Coverage criteria for logical expressions. In *Proceedings of the 14th International Symposium on Software Reliability Engineering*, Denver, CO, November 2003. IEEE Computer Society Press.
- [3] IEEE. *Standards for Software Maintenance*. IEEE Computer Society / ANSI Std 1219-1998, New York, New York, 1998.
- [4] RTCA-DO-178B. *Software considerations in airborne systems and equipment certification*, December 1992.
- [5] Y. Wu and J. Offutt. Maintaining evolving component-based software with UML. In *Seventh European Conference on Software Maintenance and Reengineering (CSMR '03)*, pages 133–142, Benevento, Italy, March 2003. IEEE Computer Society Press.