Lazy Systematic Unit Testing: *JWalk* versus *JUnit*

Anthony J H Simons and Christopher D Thomson
Department of Computer Science, University of Sheffield
{A.Simons, C.Thomson}@dcs.shef.ac.uk

Abstract

Lazy systematic unit testing with JWalk is compared against regression testing with JUnit, the leading agile testing tool. JWalk produced exhaustive test sets more quickly and recalculated full state and transition coverage, when testing modified or extended classes. For the same time and effort invested, JWalk tested up to two orders of magnitude more paths than manual tests created for JUnit by an expert tester.

1. The JWalk testing tool

JWalk is a *lazy systematic unit testing* tool [1]. The lazy systematic testing method is based on *lazy specification*, inferring a continuously changing specification from rapidly evolving code, by dynamic code analysis and programmer interaction, and *systematic testing*, generating complete test-sets that exercise and validate the state-space of the class-under-test (CUT) exhaustively to bounded depths [2].

The *JWalk* tool allows the human tester first to validate the CUT's specification by exploration, then to compile a test oracle interactively, confirming key properties of the CUT. These are re-used predictively during automated testing, which verifies the states and transitions of the CUT exhaustively.

2. JWalk challenges JUnit

A challenge was set up to contrast the effectiveness of semi-automated testing with *JWalk* against expert manual testing using *JUnit* [3], the most widely used testing tool in the agile community. The first part was to compare the coverage of expert manual test-case selection against *JWalk*'s proposed tests. The second part was to demonstrate the improved coverage of *JWalk*'s regenerated tests over regression testing.

Two related pairs of CUTs were tested, including a simple *LinkedStack*, later modified as a *BoundedStack* (a code evolution); and a *LibraryBook*, later extended as a *ReservableBook* (by inheritance). The competing testers were asked to develop "complete tests" for each initial class. Later, *JWalk* was allowed to propose further tests for the modified or extended versions.

Table 1 shows how interactive oracle confirmation in *JWalk* covered more unique cases (in less time) than the manual assertions thought up by the expert for *JUnit*. *JWalk* then automatically tested all state-transition paths to depth 3, compared against slightly less than the transition cover for *JUnit* (*nullops* were not tested; two assertions were non-unique).

Table 1. Unique tested paths

CUT	*API size*	*JUnit asserts*	*JWalk oracles*	*JWalk total*
LinkedStack	6	9	24	220
BoundedStack	7	n/a	+35	645
LibraryBook	5	11	20	138
ReservableBook	9	n/a	+167	1732

When retesting the modified or extended versions, *JWalk* found all additional observations on novel method interleavings, confirmed in under 18 minutes, and then tested up to 1732 paths automatically. *JWalk* makes better use of test automation, proposing all key test cases for rapid review by the tester, and has much higher coverage than traditional regression testing.

3. References

[1] A. J. H. Simons, "JWalk: lazy systematic unit testing", http://www.dcs.shef.ac.uk/~ajhs/jwalk/, 2007.

[2] A. J. H. Simons, "JWalk: a tool for lazy systematic testing of Java classes, by design introspection and user interaction", *J. Auto. Softw. Eng.*, 2007, to appear.

[3] K. Beck, *The JUnit Pocket Guide, 1st edn.,* O'Reilly, Beijing, 2004.