Static vs. Dynamic Testing

How Static Analysis and Run-Time Testing Can Work Together

S. Tucker Taft and Brian Lesuer

SQGNE December 2006

Outline

- The Challenges Facing Software Testing
- A Software Testing Taxonomy
- How Static Testing Can Help
- Challenges for Static Testing

SofCheck

The Challenge of Increasing Software Quality

- The Problem
  - Software testing is expensive and often incomplete, especially in the presence of third party code, open source components, cross-group integration
- The Scope
  - Software testing, qualification, integration, and reuse in a world of open source, outsourced development, mergers and acquisitions, security concerns
  - The industry is already acknowledging the growing need to supplement traditional testing and verification approaches.
  - Despite the over $1B spent on testing tools, software remains plagued by severe defects and serious security holes costing companies more than $20B to fix and costing users more than $40B in lost productivity.[1]

[1] NIST, Planning Report 02-3, May, 2002

SofCheck

3

Customers are Feeling the Pain Every Day

NASA

The first operational launch attempt of the space shuttle, whose real-time operating software consists of about 500,000 lines of code. The software error responsible for the failure, which was itself introduced when another error was fixed two years earlier, would have revealed itself, on the average, once in 67 times.

Security

In 1998, several e-mail systems suffered from a "buffer overflow error". This would occur when extremely long e-mail addresses are received. The internal buffers receiving the addresses do not check for length and allow their buffers to overflow causing the applications to crash.

Medical

In 1985, race condition errors in the inadequately tested Therac-25 software caused the deaths of 7 cancer patients due to overdoses of radiation.

Financial

In 2004, millions of bank accounts were impacted by errors due to the installation of inadequately tested software code in the transaction processing system of a major North American bank, It took two weeks to fix all the resulting errors and cost exceeded $100 million.

Telecom

In 1990, AT&T's long-distance telephone network was down for 9 hours due to an untested code patch, dramatized the vulnerability of complex computer systems everywhere. "Ghost in the Machine," Time Magazine, Jan. 29, 1990

Utilities

In 2004, a software bug was determined to be a major contributor to the 2003 Northeast blackout, the worst power system failure in North American history. The failure involved loss of electrical power to 50 million customers, forced shutdown of 100 power plants, and economic losses estimated at $6 billion.

Aerospace

In 1996, the Ariane 5 rocket exploded on its maiden flight because the navigation package was inherited from the Ariane 4 without proper testing. The overflow error that occurred caused the system to shut down and the rocket veered off course and exploded.

Gov't

In 2004, a new government welfare management system in Canada costing several hundred million dollars was unable to handle a simple benefits rate increase because its system was never tested to do so.

SofCheck

4

The Driver Behind These Issues

- Conventional software testing is expensive
  - Testing is labor intensive
    - Most QA organizations are overworked and understaffed for the volume of testing required with today's limited tools
  - Inadequate test coverage causes schedule & budget overruns
    - Research shows that traditional testing methods cover at best 70% of the code base, with up to *30%* of code *untested* as applications enter the field
    - Defects are being detected later in the cycle
    - An increasing number of these issues first manifest in the field
  - It is difficult to determine the quality of outsourced or 3rd party software
    - Undocumented limitations and programmer assumptions

SofCheck

5

A Software Testing Taxonomy

- Functional testing
  - addresses the functions and features of the application under test
  - objective is to prove that the project requirements are met by the application and that the application will fail gracefully if it receives invalid input

SofCheck

6

Testing Taxonomy

- Internationalization and localization testing
  - Ensure that the application will run under the localized version of the OS
  - Verify proper translation of strings
  - Verify that GUI labels are not truncated
  - Verify localization parameters (date, time, money)

SofCheck

7

Testing Taxonomy

- Usability Testing
  - Check Handling of Incorrect Input
  - Check Availability, Correctness, and Understandability of On-Line and Off-Line Help
  - Check Ease of Use, Number of Steps for Common Operations, etc.

SofCheck

8

Testing Taxonomy

- Security testing
  - performed to identify potential vulnerabilities in the infrastructure and transactions of an on-line application
  - Examples include:
    - obfuscation of sensitive information
    - fraud detection
    - verification that public applications do not provide entrance to non-public back-end applications

SofCheck

9

Testing Taxonomy

- Performance Testing
  - Check Speed and Responsiveness as Load Increases
  - Check Scalability as Size and/or Complexity of Input or Database Increase

SofCheck

10

Levels of Testing

- Unit testing
  - refers to testing the smallest blocks of code. This level of testing is typically performed by developers before code is checked into source control

SofCheck

11

Levels of Testing

- API Testing
  - Testing of the Application Programming Interface associated with a library of components
  - Component Usage Possibilities:
    - Directly linked into application
    - Part of a dynamically linked library (DLL)
    - Wrapped to form an Enterprise Java Bean (EJB), COM Component, POJO (Plain Old Java Object), etc. to support distributed computing
    - Available via a web service (SOAP, WSDL)

SofCheck

12

Levels of Testing

- Integration testing
  - addresses the combined parts of an application to determine if they function together correctly. The parts are the modules or components that make up the application

SofCheck

13

Levels of Testing

- System testing
  - ensures that an application functions properly with other applications in the system. It insures that the interfaces between applications work properly
  - Functional and performance is addressed

SofCheck

14

Roles for Testing

- Test-Driven Development
  - New development methodologies integrate testing into the development process
  - Write tests first, use them and enhance them along with the code they are testing

SofCheck

15

Roles for Testing

- Smoke testing
  - a subset of functional tests for a software application that represents its most important features.
  - typically very broad and very shallow
  - its objective is to provide a general indication of the health of a build as it is received from Development

SofCheck

16

Roles for Testing

- Regression testing
  - a comprehensive set of functional tests for a software application that is designed to ensure that when a change is introduced, existing features and functions are not broken.
  - the suite of tests continues to grow along with the features of the application under test

SofCheck

17

Roles for Testing

- Beta testing
  - used in some organizations as an extension of internal testing. A relatively small group of customers/users is selected and asked to “use” the software. In return the customer/user may get priority on issue resolution or first shipment

SofCheck

18

Roles for Testing

- Acceptance testing
  - is typically performed by the sponsor of a software project as a last check to ensure that the project requirements are met. This term is also used by QA groups to define the set of tests that a software application must pass in order to be deployed to production

SofCheck

19

Roles for Testing

- Post Production monitoring
  - ensures that the application functions and performs properly after deployment to production
  - provides a measurement of the testing process

SofCheck

20

White box vs. Black box

- Two different approaches to developing test cases
- Black box tests are not based on any knowledge of internal design or code
  - based on entirely on requirements, use cases and the testers intuition about how the software will be used in production

SofCheck

21

White box vs. Black box

- White box tests are based on knowledge of the internal logic of an application's code
- Tests are based on coverage of code statements, branches, paths and conditions.

SofCheck

22

Automated Dynamic Tester

- Simulates a real user interacting with the application under test (AUT)
- Essentially a robot that replaces the hands and eyes of the tester
- Provides a platform to create and run automated tests
- Typical "Real User" might actually be another program in a web-service context

SofCheck

23

Typical Dynamic Test Tool Capabilities

- Capabilities
  - Inputs data through calls that simulate the keyboard and mouse to any UI (green screen, browser, PC, Mac)
  - Verifies application results via GUI display and object properties
  - Accesses data stored in flat files or databases
  - Executes SQL statements to update or query data from ODBC compliant data sources
- Scope
  - Does not interrogate source or binary code

SofCheck

24

Benefits of Automated Dynamic Testing

- Repeatability
  - same tests get executed each time
- Productivity
  - reduces labor intensive task of manual testing
- Shorter cycles
  - tests can be executed in a shorter period of time
- Regression
  - late changes can be fully checked for regressions

SofCheck

25

How *Static Testing* Fits In

- Unit Testing
  - Tries all paths and all possible data values
- API Testing
  - Can identify pre- and post-conditions
  - Can test library without stubs or harnesses
- Smoke and Regression Testing
  - Can distinguish “old” versus “new” bugs
- Acceptance Testing for Third-Party Components
  - Can be used for overall “scorecard” without knowledge of internal workings

SofCheck

26

What Static Testing Can Detect

- Places where array indices could go out of bounds or where a null pointer might be de-referenced
- Violations of implicit preconditions of called routines
- An error-prone reliance on default initializing of numeric data
- Unsafe concurrent access to shared data
- Potential security holes

SofCheck

27

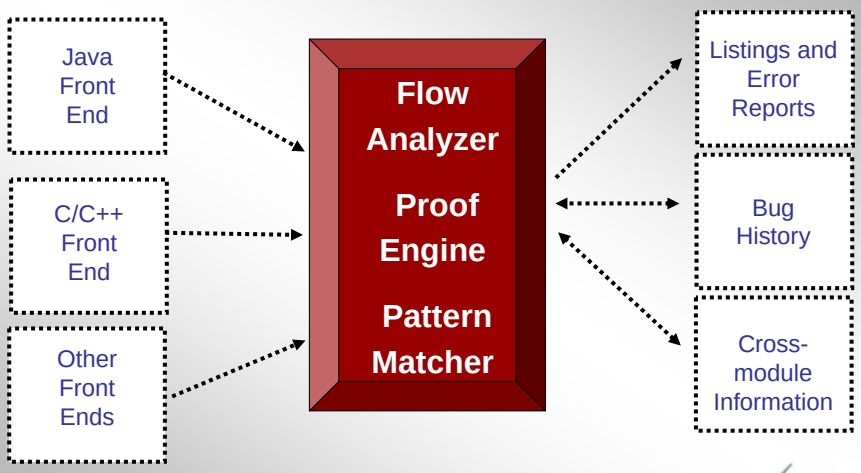
How Static Testing Can Help

- Can provide 100% full path coverage on a daily basis
- Can consider all combinations of inputs and outputs
- Can handle a user interface or an environment that is changing daily
- Can help identify the location of the root cause of the problem
- Can reduce the frustration and time-sink of debugging run-time crashes
- Can allow QA to focus more resources on functional testing

SofCheck

28

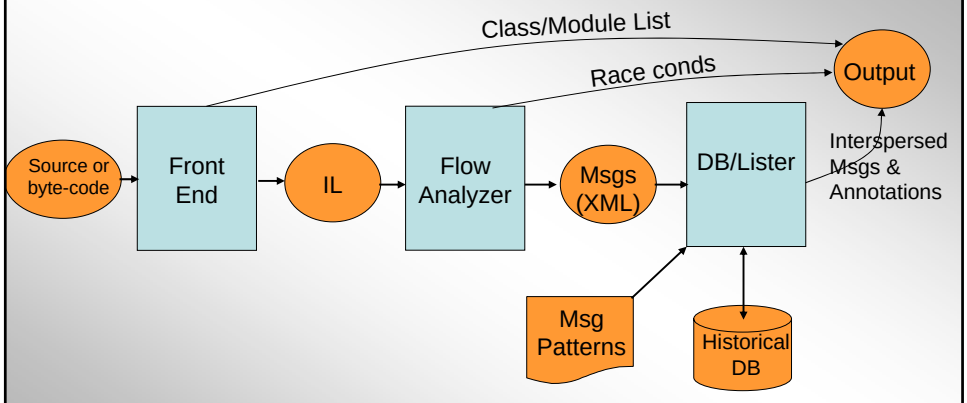
How *Static Analysis* Works



SofCheck

29

Typical Static Analyzer Phases



SofCheck

30

How Static Testing Can Help Dynamic Testing

- As a much more thorough "smoke test"
  - avoid running complete test suite if it identifies more than some threshold of “new” problems
- As part of a nightly regression test
  - Prevent “blocking bugs” from reaching QA
- To help identify parts of the code that are more "suspicious” or bug-prone than other parts
  - can apply more "conventional" QA resources to those areas
- To come up with module "test vectors”
  - help to construct run-time functionality tests that check for correct functionality for all important subranges of each input.

SofCheck

Test Vector Examples

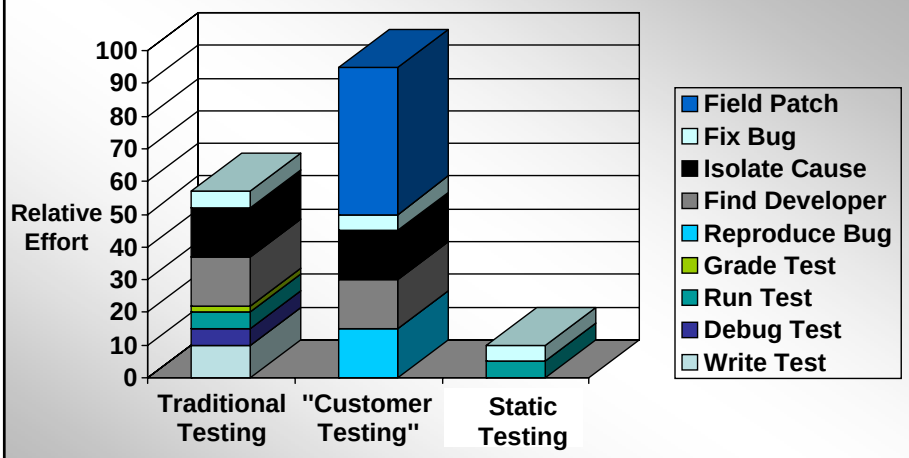
```
i2.classfile.ExceptionTable:verifyPass2
  this.catch_type: {0}, {1..65_535}
  this.end_pc - code_len: {1..65_534}, {-65_534..0}
  this.end_pc - this.start_pc: {-65_535..0}, {2..65_535}
  this.handler_pc - code_len: {0..65_534}, {-4_294_967_295..-1}
  this.start_pc - code_len: {0..2_147_549_182}, {-65_534..-1}

com.knowgate.grammar.Lexicon$Set:add
  object: {null}, Inverse{null}
  this.exclude: {0}, {1}

com.knowgate.grammar.Lexicon$Set:add
  from: {null}, Inverse{null}
  this.size: {1..4_294_967_294}, {0}
```

SofCheck

Labor Savings with Static Testing



SofCheck

33

Static Testing Sales Pitch

- In the typical development environment
  - A programmer devotes 50% of his/her time to code conceptualization and implementation
  - **50%** of his/her time is spent on testing and debugging.
- Using a median salary for a seasoned programmer of **$100,000**
  - The average programmer is being paid **$50,000** to conceptualize and implement code and **$50,000** to debug code.
- By reducing the time spent on the test/debugging process by 20%
  - Static analysis could yield a $10,000 savings/year/programmer.
  - With a team of 10 programmers, that is a **$100,000/year savings**.
- Other savings
  - QA more effective since less time lost to “blocking” bugs
  - Time to market improved
  - Fewer bugs lurking in the field

SofCheck

34

Challenges for Static Testing

- Reducing False positives (or at least prioritizing)
- Scalability (time and memory consumption)
- Providing for incremental analysis
  - small change => quick reanalysis
- Handling calls on missing code
- Properly analyzing indirect/dynamically-dispatched calls
- Allowing for user extensibility
- Analyzing for performance bottlenecks
- Analyzing all relevant languages, including scripting languages

SofCheck

35

Overall Benefits of Static Testing When Added to Dynamic Testing

- Can find places where code is questionable, or assumptions are dubious to focus QA resources
- Can look at every line of code and consider every possible input every time it runs, ensuring overall 100% test coverage
- Can eliminate “blocking bugs” before QA receives code
- Can identify pre/postconditions including indirect use of globals to help document code as written and help with functional API or component testing
- Can generate “test vectors” to help in dynamic functional testing

SofCheck

36