

Software Testing

Lecture 4 – Reviews and Evaluating Testing Methods

PART I: REVIEWS

Reviews



🔥 Kareem Carr 🔥 @kareem_carr · 21h

life in prison OR reviewing 170k lines of Matlab code. which is worse?

Accused murderer wins right to check source code of DNA testing kit used by police

New Jersey court says defendant must be able to challenge evidence

Thomas Claburn in San Francisco

Thu 4 Feb 2021 // 20:38 UTC

A New Jersey appeals court has ruled that a man accused of murder is entitled to review proprietary genetic testing software to challenge

etic sample from a weapon that was used to link the defendant to the crime.

The maker of the software, Cybergenetics, has testified in lower court proceedings that the program's source code is a trade secret. The founder of the company, Mark Perlin, is said to have argued against source code analysis by claiming that the program, consisting of 170,000 lines of MATLAB code, is so dense it would take it and a half years to review at a rate of ten lines an hour.

The company offered the defense access under tightly controlled conditions outlined in a non-

Software Verification

- “No Single technique is likely to be sufficient. Appropriate techniques / measures shall be selected according to the safety integrity level” [IEC 61508-3]
- **Reviews** provide a qualitative evaluation of correctness based on informal techniques, e.g. checklists
 - a process for verifying intellectual products by manually examining the work product, a piece at a time, typically performed by small teams of trained peers
- **Analysis** provides repeatable and detailed evidence of correctness
- Reviews and analyses do not require to execute the program
- **Testing**
 - “The process of exercising a system or system component to verify that it satisfies specified requirements and to detect errors” [DO-178]

PEER REVIEW

Definition

Peer Review

Evaluation conducted by (one or more) people with similar competencies and expertise to the author of the work

Principles of peer review

- The people most able to assess an expert's work are other experts
 - Do you agree?
- A larger and more diverse group of people are more likely to find weaknesses
- A way of providing external validation (“stamp of approval”) by community

Challenges of peer review

- It's certainly slower than checking your own work (at least, initially)
- Psychologically difficult: asking people to point out errors in your work is unnatural!
 - Do you agree?
- Can encourage:
 - Opportunism - I'll let the reviewers figure out how to fix this for me
 - Perfectionism - I must never share my work until it is flawless

Challenges to peer review

- Why are they asking me? Are they not confident in their own work?
- Are they trying to get me to do their work?
- Are they trying to show me that they're brilliant?
- Who am I to evaluate the work of a more senior expert?
- These are the wrong questions!

THE MANY TYPES OF SOFTWARE REVIEW

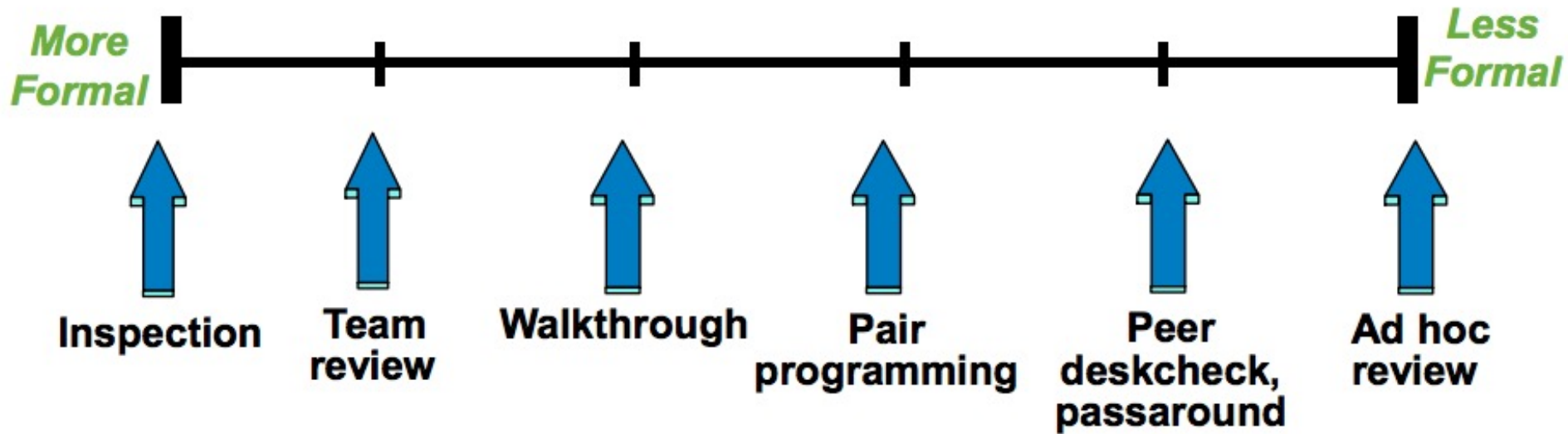
Definition

Software Review

“An evaluation of software element(s) or project status to ascertain discrepancies from planned results and to recommend improvement.”

IEEE Std 1028-1988

Types of review



Peer Reviews in Software: A Practical Guide (Wiegers)

Ad-hoc reviews

- *“Alice, are you free to help me figure out the cause of this bug for 5 minutes?”*
- Focus is on immediate problem solving

Peer desk-check / passaround

- *“Please could you take a look at pull request #243? Feedback received by Friday would be greatly appreciated!”*
- Peer deskcheck involves one person
- Passaround involves many concurrently
- Both are normally completely asynchronous & author sees only output

Pair programming

- One of the practices advocated by XP:
 - *The observer's role is to review each line of code, as it is written by the driver.*
- Has been shown to improve software quality
- But it's more a software development strategy than a peer review technique
- And it's not always a good cultural fit

Walkthrough

- A more formal meeting, typically chaired by the author
- Other developers will attend, and will expect to see a demonstration of the software
- A debugger might be used to show program state, or to justify the inclusion / exclusion of some test cases

Software Review: Walkthrough

- Goal: Evaluate software or educate an audience about a piece of software
- Principle: The author of the code guides participants and explains how it works
- IEEE 1028-1997 details 4 roles:
 - Walk-through Leader
 - Recorder
 - Author
 - Team Member

(Formal) Inspection

- Most systematic and rigorous form of review
- Carried out by inspectors, who are trained to identify common defects using checklists and analysis
- Can play a role in checking conformance to regulations and in certification
- Can also be used to collect metrics on defect rates, and to measure process improvement

Software Review: Inspection

- Goal: Systematically identify defects in the program
- Principle: the reader presents the software to the team. The outcome is *accept with no or minor corrections*, *accept with rework* (will be checked by one person), *re-inspect*
- IEEE 1028-1997 details 5 roles:
 - Inspection leader
 - Recorder
 - Reader
 - Author
 - Inspector

One possible checklist for code

- Maintainability
 - How easily could another programmer understand and modify the code?
 - Is it well-structured and adequately documented?
- Robustness
 - How will the product respond under unanticipated operating conditions? Are defaults specified for incorrect or missing inputs?

One possible checklist for code

- Reliability
 - Is it fault-tolerant?
 - Does it have effective exception-handling and error recovery mechanisms?
- Efficiency
 - How much memory or processor capacity does the program consume?
 - Are algorithms optimized and unnecessary operations avoided?

One possible checklist for code

- Reusability
 - Can components be reused in other applications?
 - Does the program have a well-partitioned, modular design with strong cohesion and loose coupling?
- Scalability
 - Can the system grow to accommodate more users, servers, data or other components?
 - Can it do so at acceptable performance and cost?

Quick quiz:

- When have you used each of the following types of review (for software, or otherwise)?
 - Ad-hoc
 - Peer desk-check / passaround
 - Pairing
 - Walkthrough
 - Formal inspection

Class Exercise

- Imagine you are to perform one or more of the following types of review:
 - Ad-hoc
 - Peer desk-check / passaround
 - Pairing
 - Walkthrough
 - Formal inspection
- How can you be a bad reviewer? How can you be a bad reviewee?

INDEPENDENCE IN REVIEW

Independence in review

- Why does independence matter?
- What does “independent” really mean?
- Why is independence not always a good thing?
- When do we need independence?

Why Independence Matters

- Political view
 - Different commercial incentives
 - Increased trust due to “second opinion” (for customer)
 - Self-protection through third-party endorsement (for developer)
- Psychological view
 - Reviewing your own work is difficult
 - Importance of “fresh eyes” and “outsider view”
- Technical view
 - Organization is a single point of failure
 - Independence builds redundancy into my system
- Systems view
 - Software coding is open loop
 - Review creates a feedback loop

Types of Independence

Are Archie and Beth Independent?

- Financial/Commercial
 - Salary of A and B come from different sources
- Organizational
 - A and B work for (report to) different people
- Task (Intellectual) Independence
 - B was not involved in the work produced by A
- Knowledge/Training
 - A and B make different assumptions
 - A and B relying on different standards and knowledge bases
- These are not YES/NO questions (see next slide)

Degrees of Independence

Consider organizational independence:

- A and B are the same person (no independence)
- A and B are different people with the same boss
- A and B work for different projects
- A and B work for different divisions
- A and B work for different companies (but their companies often work with each other)
- A and B work for different companies with a strict hands-off relationship
- A and B work for different companies in different industries

All types of independence have the same sliding scale

Trade-offs

Increased independence comes with a cost

- Financial
 - external staff, even from within the same company, are usually more expensive than using a local person
 - Indirect costs such as familiarization, contract overheads
- System knowledge
 - Typically the more independent you are, the less you know about the system and technology
- Relationships
 - Typically the more independent you are, the more you are seen as an outsider
 - Sharing of information is reduced, as is self-examination

Perfect Independence

- Can never be perfect due to financial, organizational and intellectual constraints
- Any real-world person has financial interest
 - Relationships for the purpose of repeat business
 - Liability if they do/do not support particular decisions
 - Opportunity for consultancy to fix identified problems
 - Either shared interest in project success, or competition interest in project failure
- May be an inevitable consequence of good working relationships

Undesirable Independence

- Independent review (and testing) in support of design may be a *bad* thing
 - Slight gain by having a fresh pair of eyes
 - Big loss because developer is best placed to understand the implications of failures
 - Big loss because design and verification become separate and out of sync
 - Big loss because verification becomes “over the wall” and doesn’t influence design

A QUICK EXERCISE

Swift, a Four Wheel Drive (4WD) passenger car manufacturer, has been approached by a large health organization, Save, that is interested in expanding their rapid response vehicle fleet. Rapid response typically involves a multidisciplinary team attending to a patient often in very inhospitable environments. Quick delivery time of the fleet is crucial.

Save request that Swift involve an Independent Software Assessor (ISA) in the approval process of the new fleet. Swift recruits Trust Limited, an assessment consultancy with a long established relationship with Swift. Trust has recently highlighted major safety concerns about the Electronic Stability Control software of some Swift models. These concerns proved to be incorrect, resulting in expensive and unnecessary tests. Afterwards, Trust managers went to extraordinary lengths to keep Swift happy, leading to occasions where Trust ISAs were prepared to give undue credit during safety audit sessions to assurances made by Swift engineers.

Swift senior managers have a warm feeling that they can meet the requirements for the new fleet without making any significant changes to the existing vehicle software systems. A meeting was organized at the Swift Head Office and involved: Swift (senior business manager and chief engineer), Trust: (experienced ISA), Power (engine supplier powertrain specialist) and Save (director of regional operations).

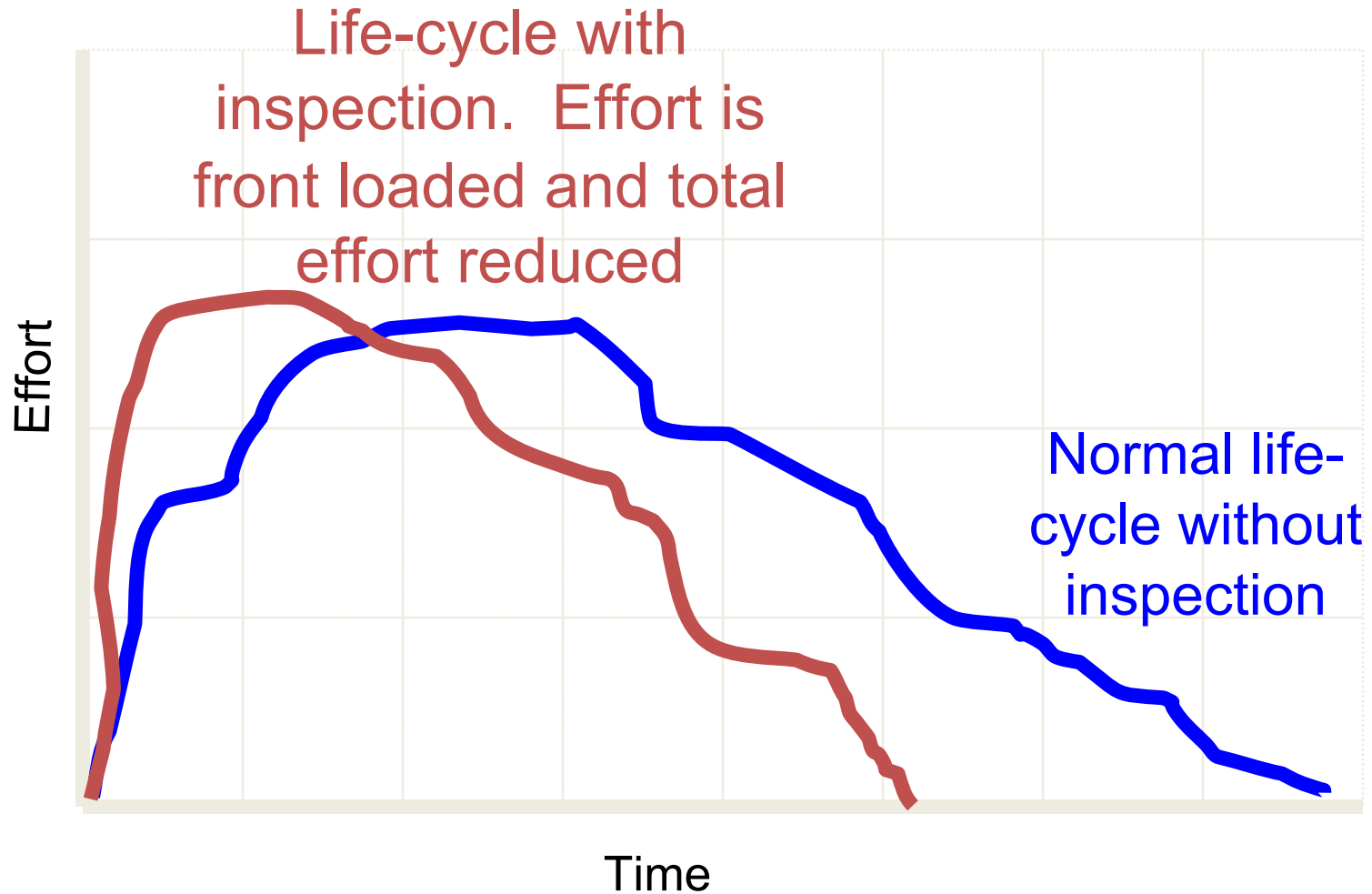
At the start of the meeting, the senior business manager from Swift praised the track record of their vehicle product lines and supported his claims by the excellent feedback they received from their customers. The chief engineer highlighted that Swift complied with best practice and were annually audited by Trust. The powertrain specialist from Power agreed with Swift about their existing record but noted his concerns about the impact that the changes in the operational profile might have on the reliability of the engines.

The senior business manager from Swift dismissed these concerns, questioning the motivation behind them (i.e. "Power test people as usual are touting for business"). He asserted that it was always possible to carry out further tests but that this might exceed the budget allocated by Save, asking the powertrain specialist to be reasonable when making any such judgments. The chief engineer from Swift added that any claims about failures to meet the targets set by Save could only be made by the Swift engineers, who were ultimately the designers of the vehicle, and the engine was one of many vehicle components. The director of regional operations from Save explained that he was not an expert in vehicle safety and turned to the ISA for advice on the best course of action.

How should the ISA respond?

HOW TO MAKE THE MOST OF (SOFTWARE) REVIEWS

Reviews should frontload effort



Defects – detect and prevent

- Of course, we would like to detect more errors earlier by performing better reviews - “Product inspection”

When to do what?

“Use walkthroughs for training, reviews for consensus, but use inspections to improve the quality of the document and its process.”

Tom Gilb

PART II – EVALUATING TESTING METHODS

Code

Design

Architecture

Requirements

Documentation

Tests

Recap – what goals might you have when testing?

- Find the maximum number of bugs
- Know whether you have undiscovered bugs
- Comply with regulator-set standards
- Have a compelling defence in a court case
- Do any of the above in the minimum time and cost
- ...

Recap – coverage metrics

- Statement
- Branch
- Path

- Condition
- Combinatorial
- MC/DC

- But we know that they aren't perfect...

And it's even worse than we thought

“While coverage measures are useful for identifying under-tested parts of a program, and low coverage may indicate that a test suite is inadequate, **high coverage does not indicate that a test suite is effective.**”

Coverage Is Not Strongly Correlated with Test Suite Effectiveness,

Inozemtseva and Holmes., in Proc. ICSE, 2014.

See also The Effect of Program and Model Structure on MC/DC Test Adequacy Coverage,

Rajan et al., in Proc. ASE, 2008.

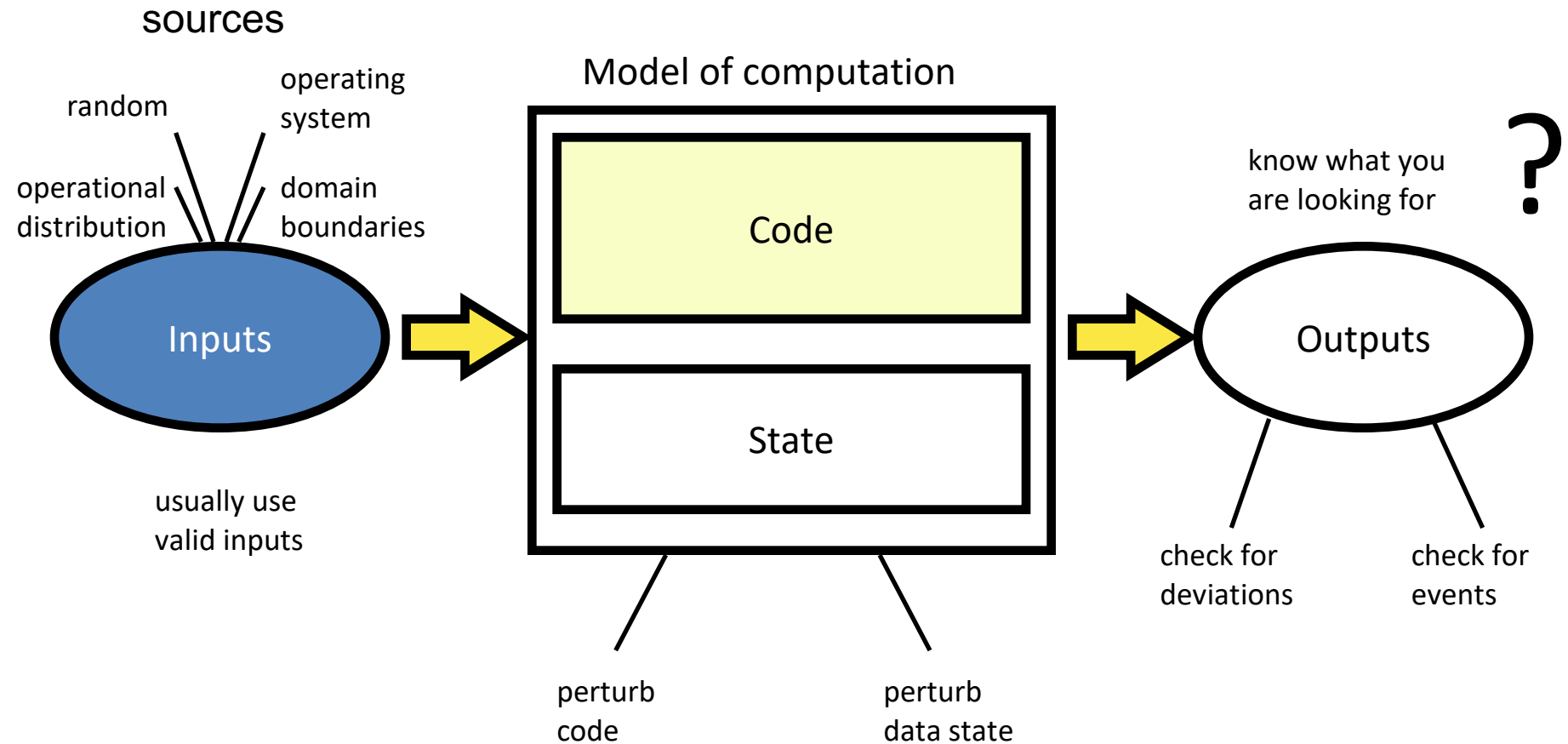
How do we know if tests are detecting real faults?

- One method:

Fault Injection

Testing **our tests** by deliberately introducing faults into the software **and checking to see if we noticed.**

Software Fault Injection



Fault injection is the modifying of inputs, code or state data to see what happens at the outputs

Fault injection: non-traditional testing

- Fault injection is incapable of demonstrating correctness
 - Because the program is run in an altered state
- Fault injection is capable of demonstrating what sort of outputs the program produces under anomalous circumstances
 - Can be important for safety to see how the software behaves in anomalous circumstances
 - For safety-critical systems: whether additional safety requirements are needed
 - A means for inoculating the code against the effects of anomalies
 - Related to robustness and stress testing

Quis custodiet ipsos custodes?

Who will guard the guards themselves?

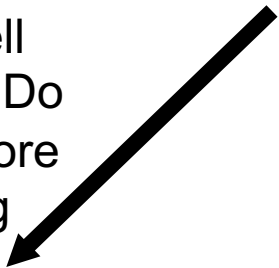
Effective Test Cases

- How do I know if my test cases are effective enough to find errors?
- How can confidence in the effectiveness of these test cases be estimated?
 - Key to demonstrating trustworthiness of V&V evidence
- Traditionally, the quality of test cases is estimated through peer review and coverage metrics of the software under test

Testing the Test Data

How well have you
tested your program?

Not well
enough. Do
some more
testing



But what extra do
you need to do?

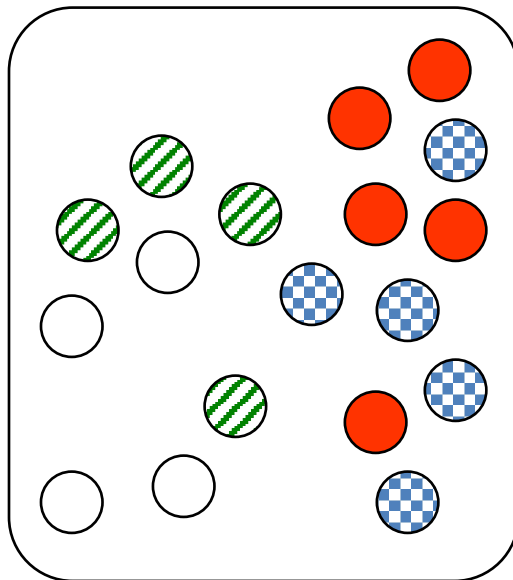
Well enough.
Stop!



But how can you
tell?

Error Seeding

- Error seeding is early example of fault injection:
 - plant bugs and see how many are caught
 - hypothesis: proportions of *planted* and *actual* bugs discovered are the same



- Planted bug found
- Planted bug not found
- Actual bug found
- Actual bug remaining?

$$\frac{\# \text{ } \text{●} }{\# \text{ } \text{●} + \# \text{ } \text{●} } = \frac{\# \text{ } \text{●} }{\# \text{ } \text{○} + \# \text{ } \text{●} } \quad ?$$

means “number of”

Mutation testing uses the program to test the test data

It **assesses** the ability of **your test set** to distinguish between the original program and programs that differ from that original in a single small way.

These small variants are called ***mutants***.

The more mutants distinguished from the original the better.

Mutation Testing

- Concerned with programmer-induced faults
 - i.e. programming slips
- **Process:**
 - assume program **is roughly** correct
 - “**competent programmer hypothesis**”
 - create mutant programs **by tweaking various syntactic elements**
 - **e.g. change + to -**
 - execute test cases **to see if each mutant’s behaviour differs from original in at least one test case**
 - if so, it is said to have been *killed*
 - **derive *mutation score*: proportion of mutants killed**
 - **How might you define mutation score?**
 - **devise further tests to increase the mutation score**
 - **Why might the mutation score be less than 1.0?**

Mutation Operators – Examples

- Operand replacement
 - Variable initialization elimination
 - Constant replaced by a variable
 - Array name replaced by another array name
 - Pointer name replaced by another pointer name
- Statement modification operators
 - Replace statements within loops with operators
 - Replace WHILE with repeat-until
- Expression modification operators
 - Replace each unary operator (+, -, ABS) with each other unary operator
 - Insert the unary operator ABS in front of every arithmetic expression

Types of Mutants

- “Do Fewer” optimization
 - Mutant Sampling
 - Constrained Mutation
 - Mutant Clustering
- “Do Faster” optimization
 - Schema based Mutation
 - Separate Compilation Approach
 - Runtime Optimization Techniques
- “Do Smarter” optimization
 - Human Error vs. Completeness
 - Novel Computer Architectures

Industrial Perspective

- Mutation testing has been perceived as too expensive
 - Due to the high computation and manpower overheads in developing a mutant set, re-executing test sets and then analysing the results
- There are increasing developments towards automation
 - Combined with greater computational performance.
- There is still a lack of support for safety-critical systems
 - Traditionally developed in languages which provide an inherent level of assurance in their design (Ada for example)
- Mutation testing tools have tended to focus on languages which are commercially in demand (Java)

Richard Baker & Ibrahim Habli

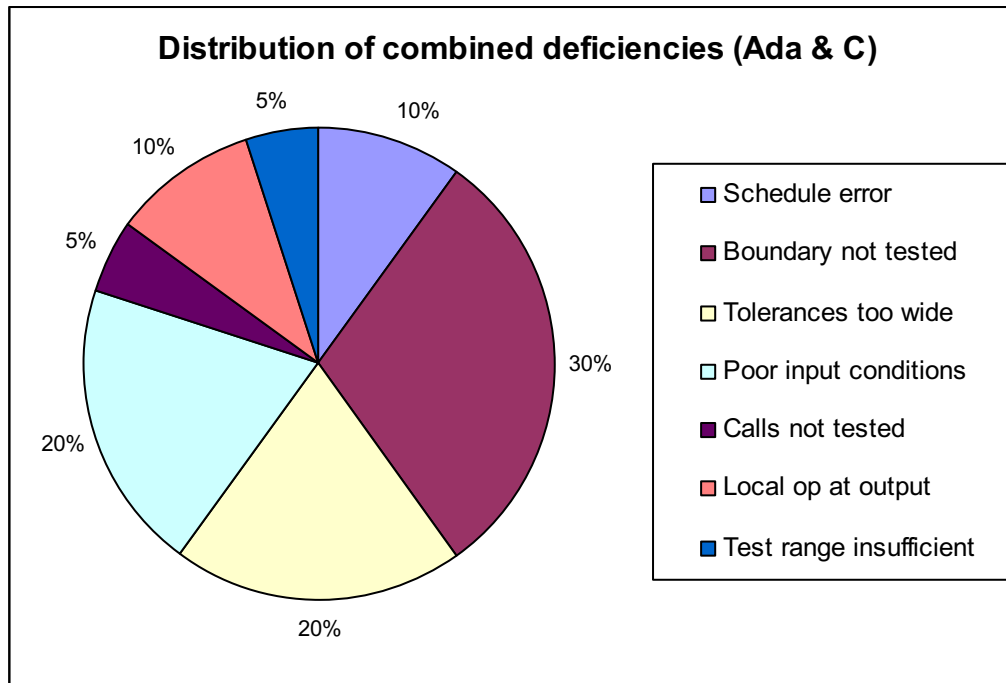
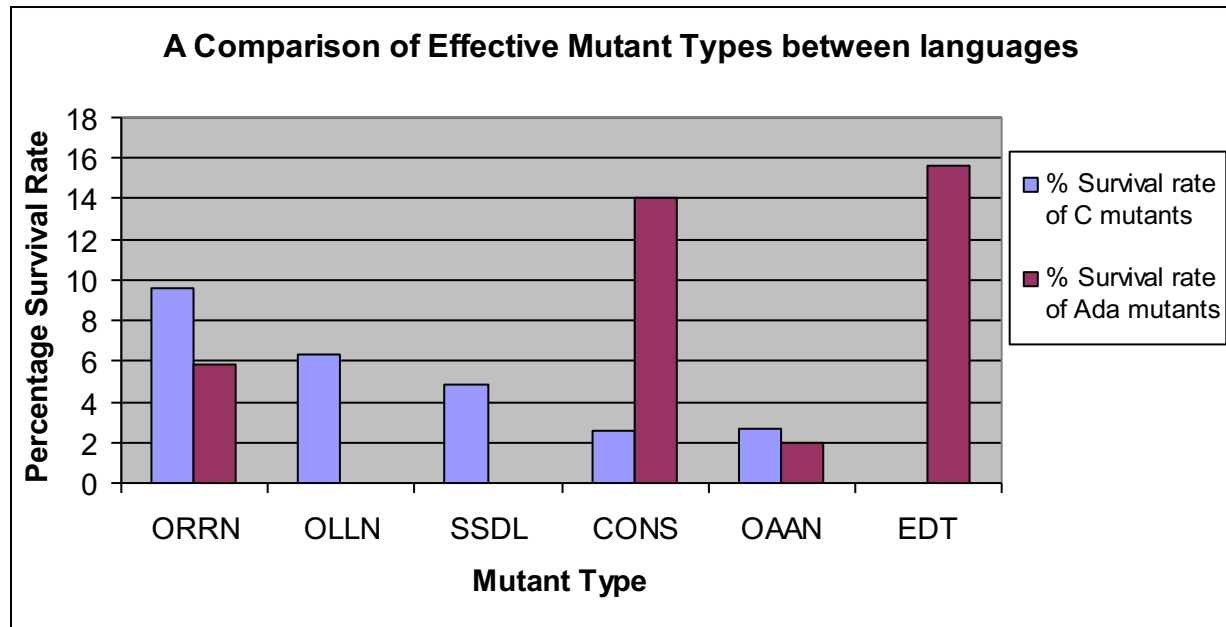
IEEE Transactions on Software Engineering

AN EMPIRICAL EVALUATION OF MUTATION TESTING FOR IMPROVING THE TEST QUALITY OF SAFETY-CRITICAL SOFTWARE

Study

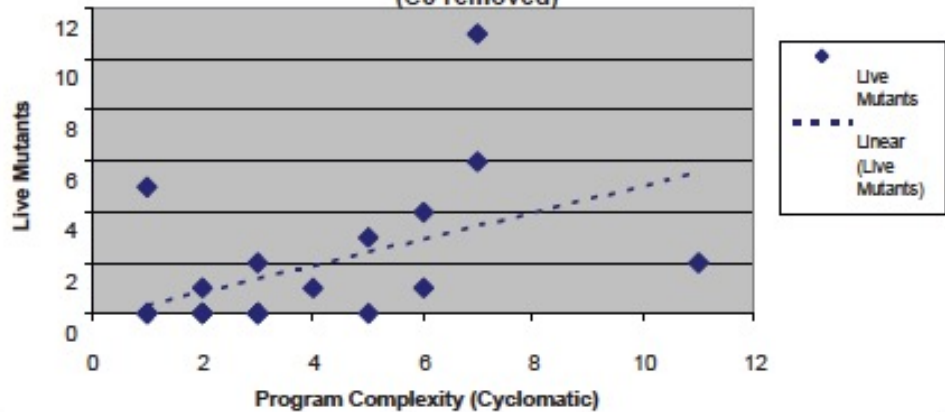
- Two engine control and monitoring software systems developed in SPARK Ada and MISRA C
 - Already met certification requirements levels A & C
 - E.g. MC/DC and statement test coverage
- Generated 3,149 mutants for C program and 651 mutants for Ada program
 - 8 of 25 Ada code items failed to achieve 100% score
 - 11 of 22 C code items failed to achieve 100% score

Further Examination

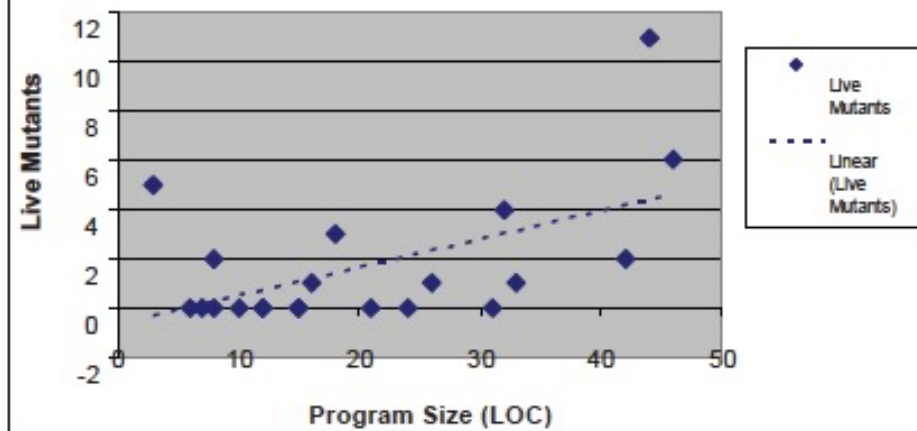


Survival Rates vs. Code Size and Complexity

Live Mutants vs Program Complexity
(C8 removed)



Live Mutants vs Program Size (C8 removed)



Study Conclusions

- Test engineers are too focused on coverage targets and less focused on producing well designed test cases
- A cyclomatic complexity score ≥ 3 was enough to identify deficiencies in test cases
- Mutation testing could be useful where traditional test coverage methods might fail

Combining V&V Evidence

- Criteria for combining evidence
 - to address different causes of failures
 - e.g. timing, incorrect inputs or incorrect processing
 - to try to compensate for limitations in the techniques
 - can also think of this as giving diversity, and reducing effects of errors in techniques and tools
- In practice, will require a large number of techniques
 - to be useful, need to be able to isolate small parts of code to focus techniques
 - e.g. use information flow analysis to show modules independent
 - then do, say, black box testing with fault injection

- Any questions?