Reviews and Evaluation Methods SFWR ENG 3S03: Software Testing

A. Marinache

Department of Computing and Software, McMaster University
Canada L8S 4L7, Hamilton, Ontario

Acknowledgments: Slides adapted from Dr.R.Paige

Objectives

- Understand why we need software reviews
- Explore different software review types
- Explore evaluation methods in more detail

- Why bother (again)?

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: a qualitative evaluation of correctness based on informal techniques
- Analysis: repeatable and detailed evidence of correctness
- Testing "The process of exercising a system or system component to verify that it satisfies specified requirements and to detect errors" [DO-178]
- Manual/Automatic? Static/Dynamic?

Feature	Review	Analysis
Method	Manual	Automated (Static/Dynamic)
Focus	Logical correctness, readability, maintainability	Performance, security, defects
Who	Developers, testers, stakeholders	Automated tools
Scope	Documents, code, designs, test cases	Source code or running application
When	Before development progresses	During development/testing
Outcome	Recommendations for improvement	Defect reports, security findings

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

- Objectivity & Constructive Feedback
  - Encourages team collaboration
- Well-Defined Process
  - Ensures reviews are systematic and effective
- Review Small and Incremental Changes
  - Improves focus and accuracy
- Use Checklists
  - Prevents missing critical issues
- Keep Reviews Time-Bound
  - Reduces reviewer fatigue

Principles of Peer Review (contd)

- Use Automated Tools Where Possible
  - Focus on Logic, design, not syntax
  - Reduces manual errors and speeds up review
- Ensure Participation from Multiple Reviewers
  - Provides diverse insights
- Foster a Culture of Continuous Improvement
  - Encourages participation and learning
- Require Author Involvement in Review
  - Ensures discussions and proper understanding
- Track Review Metrics & Improvements
  - Helps improve future reviews

Challenges of Peer Review

- Slower initially than checking your own work
- Psychologically difficult
- Can encourage opportunism & perfectionism

Challenges to Peer Review

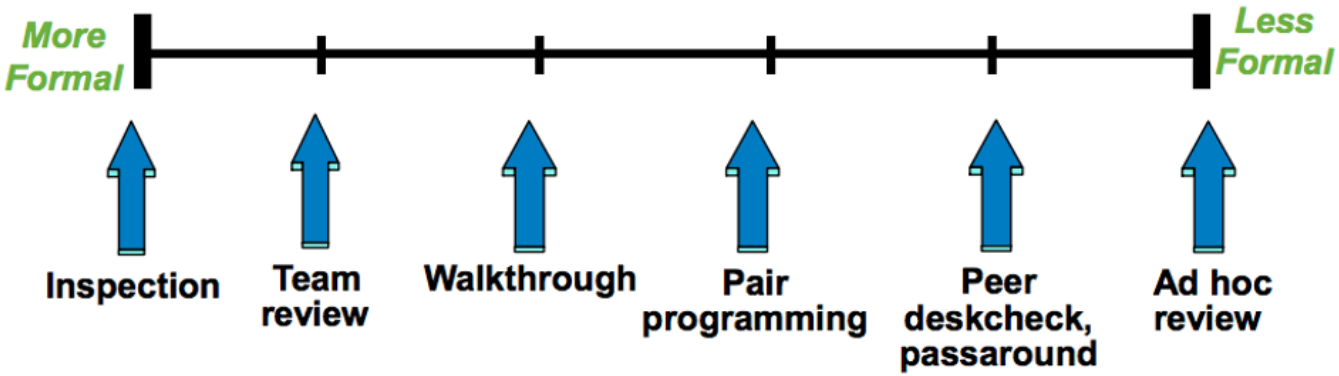
- Why are they asking for review (are they not confident in their own work)?
- Are they trying to get me to do their work?
- Are they trying to show off?
- Who am I to evaluate the work of others?

Wrong or right questions?

Definition (Software Review)

An evaluation of software artefacts (or project status) to ascertain discrepancies from planned results and to recommend improvement

Source: IEEE Std 1028-1988



- Ad-hoc reviews (informal)
  - "John, are you free to help me figure out the cause of this bug for 5 minutes?"
  - Focus is on immediate problem solving

- Peer deskcheck/passaround
  - "Please could you take a look at pull request #243? Feedback received by Friday would be greatly appreciated!"
  - Focus is on minimal peer review
  - Peer deskcheck involves one person
  - Passaround involves many concurrently
  - Both are asynchronous & author sees only output

- Pair programming
  - XP practice: the observer's role is to review each line of code as it is drivven by the driver
  - Focus is on real time review and collaboratioon
  - Is shown to improve software quality
  - May not be a good cultural fit

- Walkthrough
  - More formal meeting, in which the author guides the participants and explains how the code works
  - Focus is evaluating sofwtware and sharing knowledge (educational process)
  - IEEE 1028-1997 details 4 roles
    - Walkthrough leader
    - Recorder
    - Author
    - Team Member

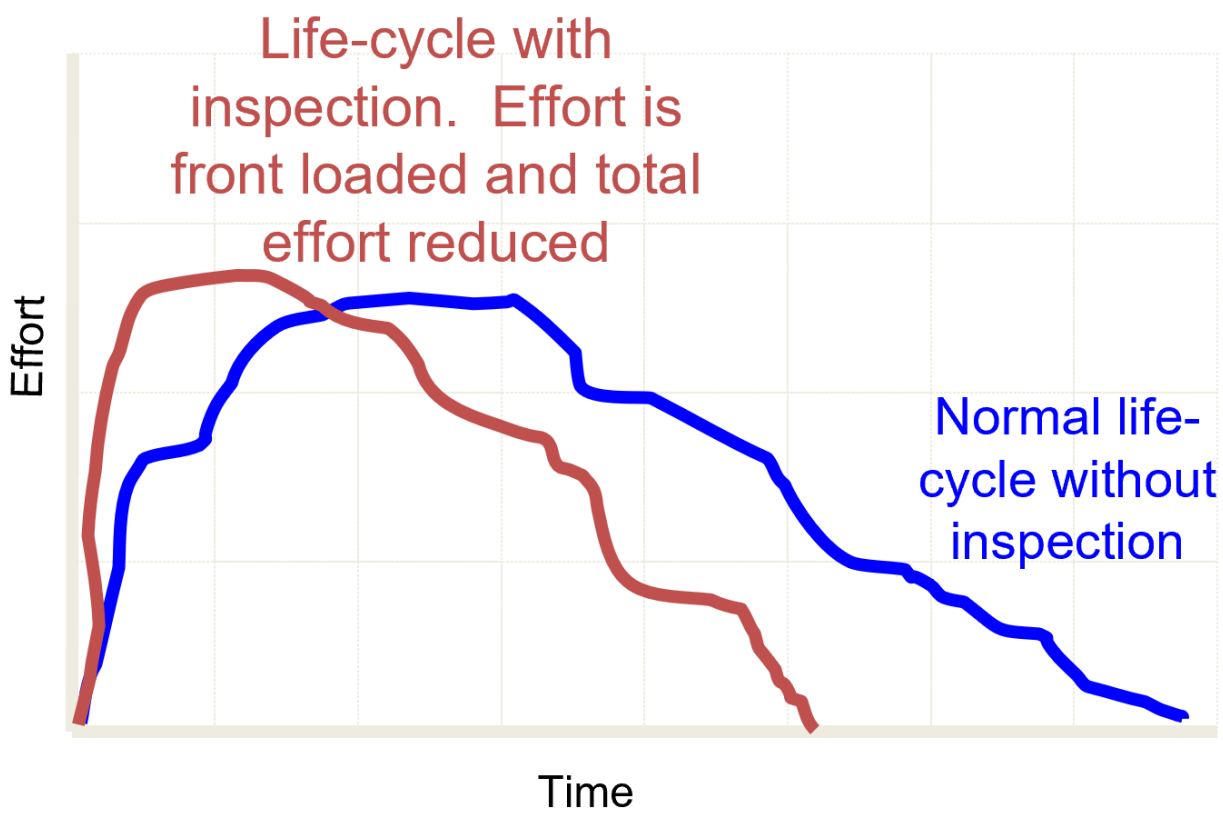
- (Formal) Inspection
  - Most systematic and rigorous form
  - Focus is on formal, systematic defect detection
  - Inspector identify commond defects using checklists & analysis
  - Regulator and Certification may require it
  - Can be used to collect metrics on testing process (defect rates, process improvement)

- (Formal) Inspection
  - ISO 1028-1997 details 5 roles:
    - Inspection Leader
    - Recorder
    - Reader
    - Author
    - Inspector
  - Reader presents the software. Outcome
    - accept with minor or no corrections
    - accept with rework
    - re-inspect

- Maintainability
  - Is it well structured & documented?
  - How well could another developer understand and modify it?
- Robustness
  - Are defaults specified for incorrect or missing inputs?
  - How will the system respond to unanticipated operating conditions?

- Reliability
  - Is it fault-tolerant?
  - Does it have effective exceptiuon handling & error recovery?
- Efficiency
  - How much memory or processor capacity does it consume?
  - Are algorithms optimized and unnecessary operatios avoided?

- Reusability
  - Can components be reused in other applicatioons?
  - Does it have modular design, strong cohesion, low coupling?
- Scalability
  - Can the system grow to accommodate more users, data, components, etc.?
  - Can it do so at acceptable perfromance and cost?



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

Tom Gilb

- Imagine you are in one (or more) of the review types
  - Ad-hoc
  - Peer deskcheck/passaround
  - Pair Programming
  - Walkthrough
  - Inspection
- How can you be a bad reviewer?
- How can you be a bad reviewee?

Why Independence Matters

- Political view
  - Company POV: different commercial incentives
  - Customer POV: increased trust
  - Developer POV: self-protection (through 3rd party endorsement)
- Psychological view
  - Revieweing your own work is difficult
  - "Outside view" is important to counter bias
- Technical view
  - Eliminate single point of failure
  - Build redundancy into the system
- System view
  - Building the system is open loop
  - Review creates a feedback loop

Types of Independence (for **A**rchie and **B**eth)

- Financial/Commercial
  - Salaray of A and B come from different sources
- Organizational
  - A and B report to different people
- Task/Intellectual
  - B was not involved in the work produced by A
- Knowledge/Training
  - A and B make different assumptions
  - A and B rely on different standards/knowledge bases

Degrees of independence (Organizational)

- A and B are the same person
- A and B have the same boss
- A and B work for different projects
- A and B work for different divisions
- A and B work for different companies (companies work with each other)
- A and B work for different companies (same industry, strict hands-off relationship)
- A and B work for different companies in different industries

Cost of Independence

- Financial
  - External staff
  - Indirect costs familiarization, contract overheads
- Technical
  - Independent parties know less about the system and technology
- Relationship
  - Outsider syndrome

"Perfect" Independence

- Cannot be achieved!
- Humans have a personal (financial) interest
  - Relationship for future business
  - Liability if they do/do not support decision(s)
  - Opportunity for consultancy to fix identified issues
  - Shared interest in project success
  - Competiton interest in project failure
  - etc.

When Independence may be a bad thing for desing activities

- Slight gain by having a fresh pair of eyes
- Big loss: author understands best the implication of failures
- Big loss: design and verification become out of sync
- Big loss: verification is external, doesn't influence design

- **Swift**, a 4WD car manufacturer, is approached by **Save**, a health organization, to provide vehicles for rapid response teams
- Save insists on involving an **Independent Software Assessor (ISA)** due to safety concerns
- Swift recruits **Trust Limited**, a consultancy they have worked with in the past, but Trust has previously made false safety claims, which created tension
- Swift's senior management believes the current vehicle software is sufficient and dismisses safety concerns raised by **Power**, their engine supplier, citing costs
- Save's director of operations, not being a technical expert, turns to the ISA for guidance during the meeting

- How **should** the ISA respond? How do you think they will respond?

- Requirements
- Documentation
- Architecture
- Design
- Code
- Tests

Recap: What goals we might have when testing?

- Find (and fix!) maximum number of bugs
- Know if we have undiscovered bugs
- Comply with regulator-set standards
- Have a compelling defence in a court case
- Do the testing with minimum time and cost
- ...

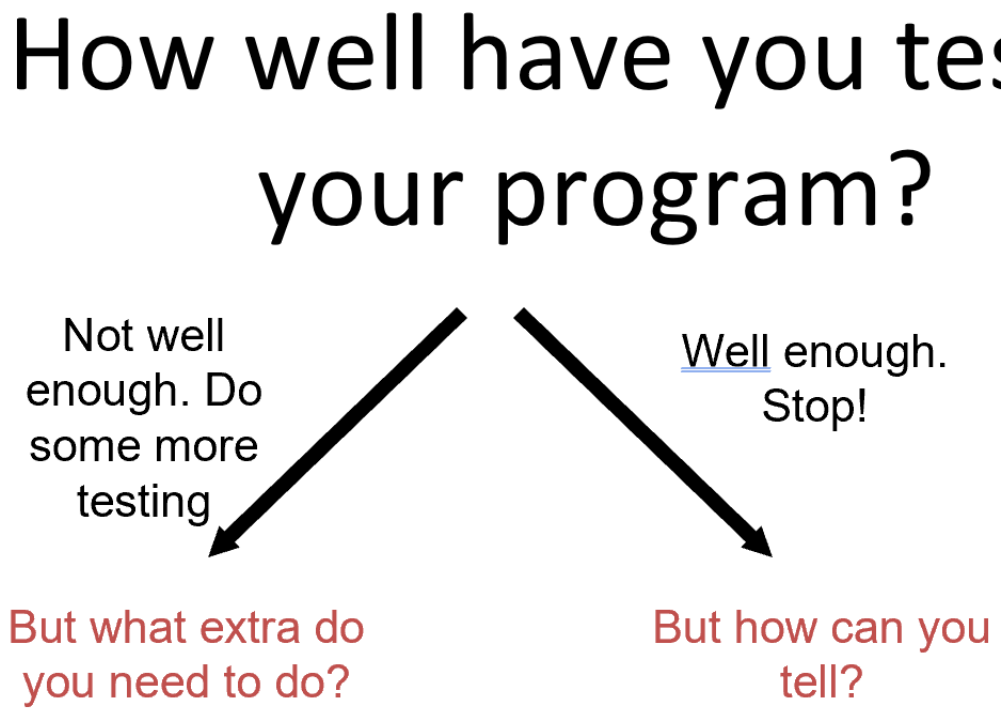
Recap: Coverage Metrics

- Statement
- Branch
- Path
- Condition
- MCC
- MC/DC

But...

"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." [IH14]

See also: The effect of program and model structure on MC/DC test adequacy coverage [RWH08]

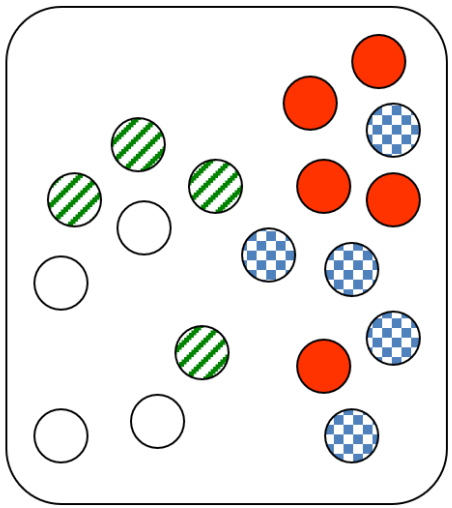


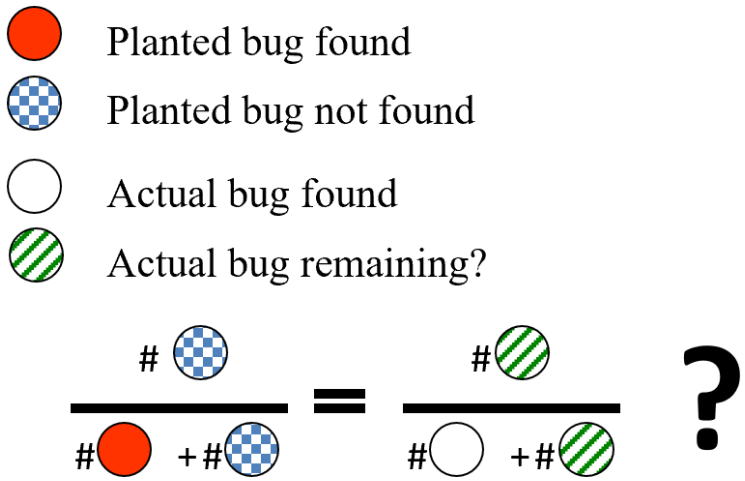
Definition (Fault Injection)

Testing the tests by deliberately introducing faults into the software and checking if we notice

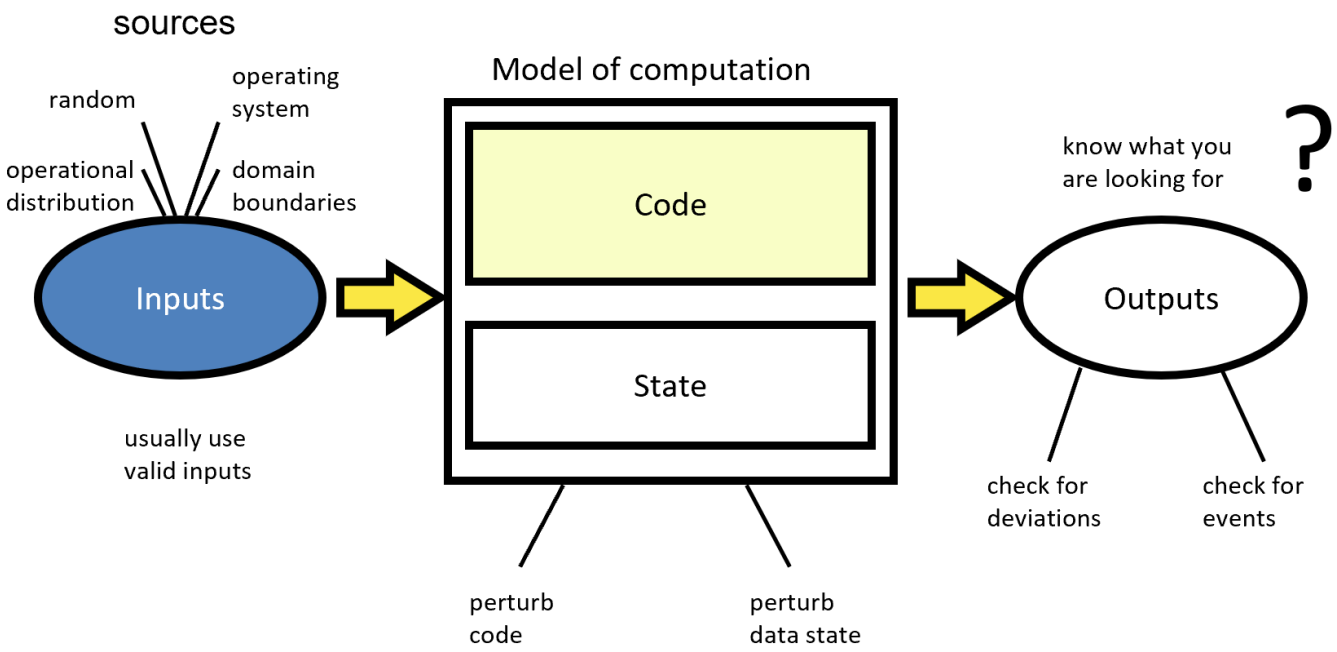
- Different focus: we evaluate the test set(s)

- Error seeding as FI
- Process: Plant defects and see how many are caught
- Hypothesis: Proportions of planted and actual defects discovered are the same





Fault Injection Process



- FI cannot demonstrate correctness
- FI can show what happens under anomalous circumstances
  - Safety critical systems: are additional safety requirements needed?
  - A means of "inoculating" the system against the effects of anomalies
  - Used for: Robustness and stress testing

Quis custodies ipsos custodes?

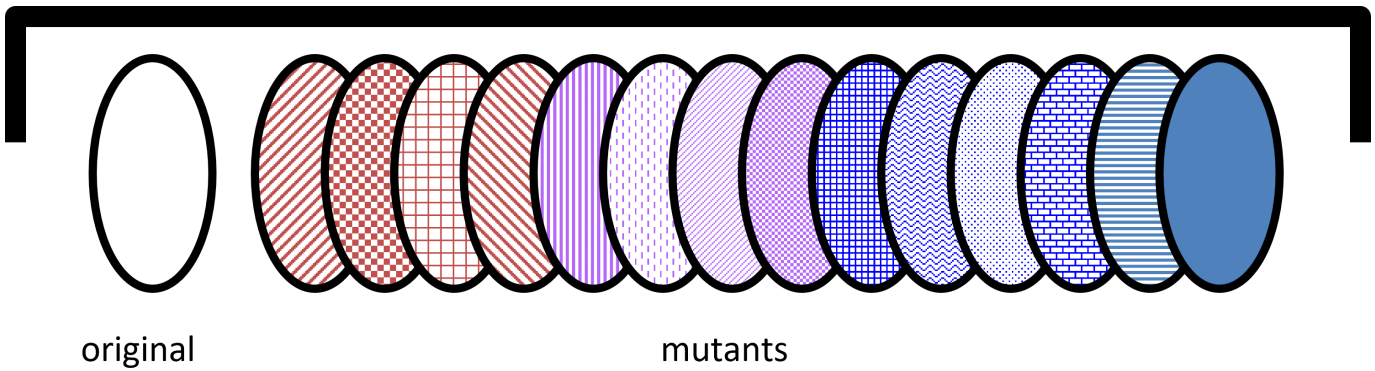
Who guards the guards themselves?

- 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 model evidence
- Traditionally, the quality of test cases is estimated through peer review and coverage metrics of the software under test

- Mutation testing uses the program to test the test data
- It assesses the ability of the test set to distinguish between the original system and one that differ from the original in a single small way
- The variants are called mutatnts
- Evaluation: the more mutants are distinguished, the better

- 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
  - The mutant is said to have been killed
- Derive mutation score: proportion of mutants killed
- Devise further tests to increase the mutation score
  - Why might the mutation score be less than 1.0?

- Can also have the actual program and all mutants in separate modules
- Simple but costly in terms of space and run-time
- Typically, large programs will cause major problems



- 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

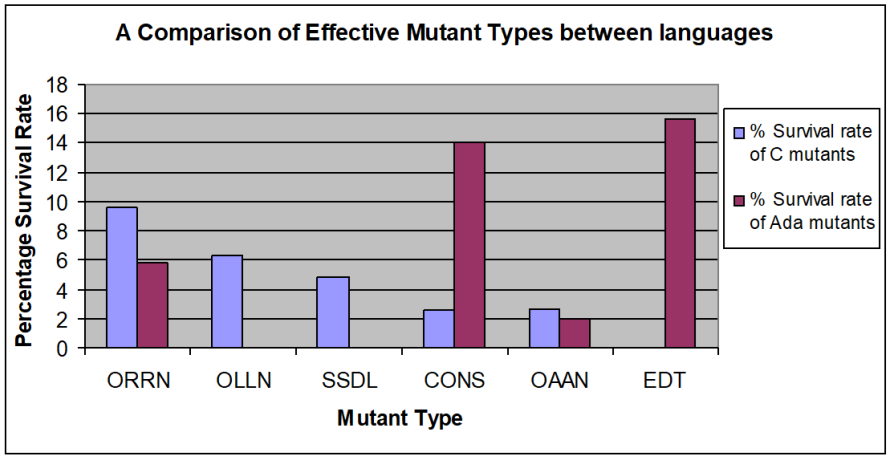
Industrial Perspective

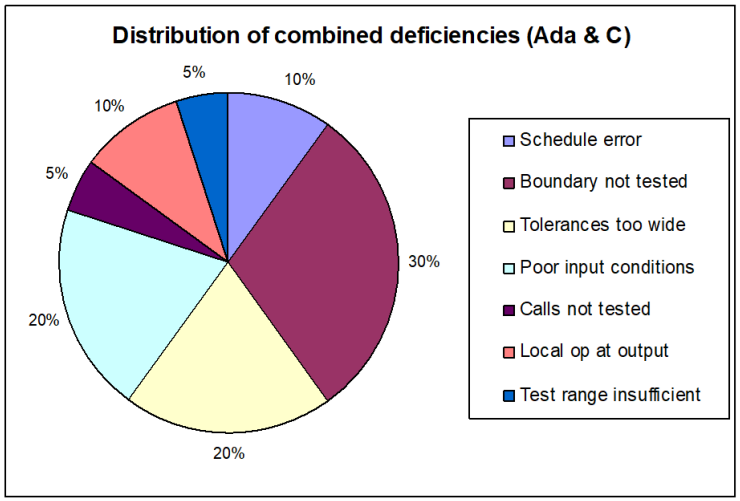
- Mutation testing has been perceived as too expensive
- There are increasing developments towards automation
- There is still a lack of support for safety-critical systems
- Mutation testing tends to focus on languages that are commerchially in demand

Study: An empirical evaluation of mutation testing for improving the test quality of safety-critical software [BH12]

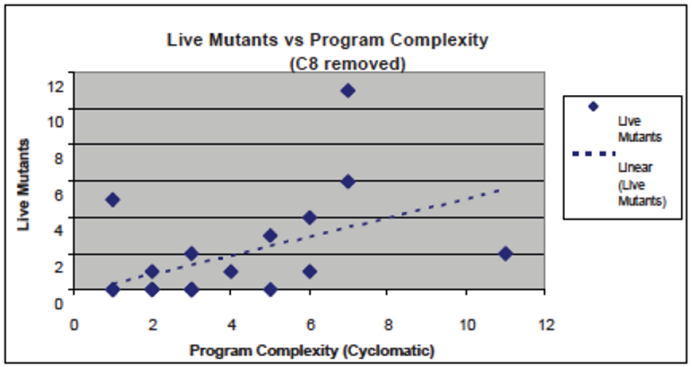
- Two engine control and monitoring software systems developed in SPARK Ada and MISRA C
  - Already met certification requirements levels A & C (MC/DC)
- Generated 3,149 mutants for C program and 651 mutants for Ada program
  - 8 of 25 Ada code items failed to achieve 100% score
  - 11 of 22 C code items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

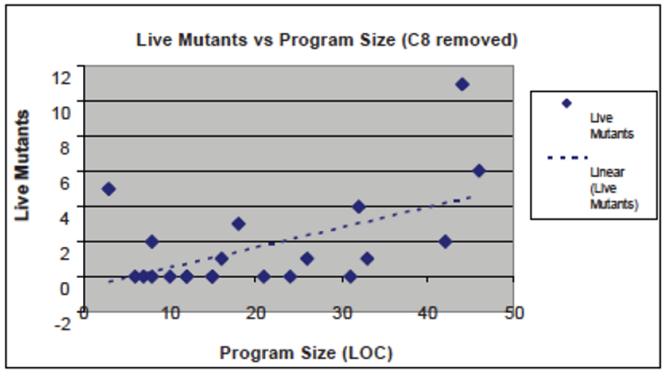
Further Examination





Survival Rates vs. Code Size and Complexity





- Measurement is hard!
- Measurement applied to code is really hard!
- What are you actually measuring?
  - The code?
  - The code's behaviour?
  - The code's architecture?
  - The code's complexity???

Study Conclusions

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

Combining V-model 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
- Will require a large number of techniques (in practice)
  - 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 black box testing with fault injection

References I


Richard Baker and Ibrahim Habli, An empirical evaluation of mutation testing for improving the test quality of safety-critical software, IEEE Transactions on Software Engineering **39** (2012), no. 6, 787–805.


Laura Inozemtseva and Reid Holmes, Coverage is not strongly correlated with test suite effectiveness, Proceedings of the 36th international conference on software engineering, 2014, pp. 435–445.


Ajitha Rajan, Michael W Whalen, and Mats PE Heimdahl, The effect of program and model structure on mc/dc test adequacy coverage, Proceedings of the 30th International Conference on Software engineering, 2008, pp. 161–170.