Software Engineering: Theory and Practice

Engineering and Errors

Verification by Review

Tom Verhoeff

Eindhoven University of Technology
Department of Mathematics & Computer Science
Software Engineering & Technology

Feedback to `T.Verhoeff@TUE.NL`

What Is an Error?

- What is the most "impressive" error that you have made?

  Watch video of Ariane 501 flight.

- Beautiful versus ugly: an opinion, not an error

Terminology: IEEE Classification

- Failure: product deviates from requirements during use/operation
- Defect, fault: anomaly in a product that can somehow (eventually) lead to a *failure*
- Mistake: human action ("slip") causing a *fault*
- Error: difference between actual and specified/expected result

Assumes requirements/specification/contract (establish in advance)

Economy of Defects

- The longer a defect is undiscovered, the higher its cost: grows exponentially in distance between injection and removal.
- Defects decrease the predictability of a project: cost (time) of defect localization and repair is extremely variable.
- Defects concern risks, i.e. uncertainty; product could be defect-free at once, but defects are likely.
- The likelihood of defects increases rapidly with higher system complexity.

Dealing with Defects

1. Admit that people make mistakes and inject defects
2. Prevent them as much as possible
3. Minimize their consequences (fault tolerance)
4. Detect their presence as early as possible
5. Localize them
6. Repair them
7. Trace them: find root causes and possible other consequences
8. Learn from them: improve the process and tools

Awareness

"Programmers [Engineers] will always make errors. No advance in formal [methods] will ... prevail over human fallibility.

[T]here are two approaches to software errors:

- one accepts them as inevitable and steers work toward removing faults that errors produce;
- the other ignores errors, the resulting faults, and the failures they may cause, and replaces testing, discovery, and repair with legal and business maneuvers."

Robert N. Britcher. *The Limits of Software*. Addison-Wesley, 1999.

Preventing Defects (or instant detection and repair)

- Impossible to do for 100%, but prevention offers the biggest gains
- Every defect not prevented adds work (cost)
- Remove sources for mistakes (e.g., improve syntax of prog. lang.)
- Always work neatly, also on prototypes, test software, ...
- Use checklists and standards
- Work in pairs
- "Think before you act"

Detecting Defects

- Reviewing:
  - Examine an artifact *with the intent of finding defects*.
  - Can be done early in the development process.
  - Often localizes the defects as well.
  - Can and should also be applied to code.
- Testing:
  - Use a product systematically *with the intent of finding defects*.
  - Works through failures; does not localize underlying defects.
  - Requires a working product (part).

Limits of Testing

Edsger W. Dijkstra (CACM, 1972):

> "Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence."

Testing in itself does not create quality.

Dijkstra's advice: Prove mathematically that an artifact has required properties. Ideally: let proof development drive the design, leading to Correctness by Construction.

Removing Defects

Debugging: localize, diagnose, and correct detected defects

Time consuming and unpredictable process

Coding Standards

- Restrict what program code "looks" like
- Layout: indentation, spacing, blank lines, line length; at most one definition/declaration/statement per line
- Naming: constant, variable, method, class, attribute
- Comments: file header, "contract" (assumption, effect), explain variable declaration or statement
- Structure: how to order things; maximum size of code blocks

Why Use Coding Standards?

- You make fewer mistakes.
- If you make them, they are found more easily and more quickly.
- If you cannot find them yourself, then others can help you more effectively.
- In case of law suits, you are in a better position to defend yourself.

Costs of Dealing with Defects Responsibly

- Standardization (e.g. of coding style), reviewing, testing, ... all cost extra effort and time (mostly initially).
- Consider this to be a small pre-paid insurance fee.
- Not using these techniques increases risks and unpredictability, and hence increases costs, often considerably.

Why Do Reviewing?

- Early detection of deficiencies and risks:
  - Humans make mistakes, no matter what.
  - Late detection is (very) costly.
  - Testing cannot find all (kinds of) defects.
- Finding defects directly, rather than detecting failures (by testing).
- Communication of knowledge
- Monitoring of status and progress

Types of Reviewing

- Management review: of a process, on behalf of managment
- Technical review: of a product (not of its creator)
- Inspection: visual examination, by peers
- Walk-through: creator explains artifact to others
- Audit: independent examination, also of an organization

How to Do Reviewing

1. Determine type, purpose, and timing of review in advance.
2. Check initial fitness (e.g.: is document spell checked).
3. Select and inform reviewers (and train them, if needed).
4. Distribute material on time.
5. Prepare individually (read material, make notes).
6. Hold meeting (pre-determined roles), decide on recommendations.
7. Write and present a review report.

General Advice on Reviewing

- Take reviews seriously and spend time well.

  Do not waste time on trivialities.

- Use checklists and standards.

- Stick to the purpose (e.g. do not criticize creators).

- Do not try to solve problems while reviewing.

  But: do recommend changes, also to the development process.

- React on review outcome (do rework, adjust the process).

References

- *Ariane 5 Failure: Full Report* by ESA
- "The $100,000 Keying Error", *IEEE Computer*, pp.106–108, April 2008.
- *Code Conventions for the Java Programming Language* by SUN
- *Java Coding Standards* by ESA