Software Engineering: Theory and Practice

NMA – National Student Academy
Summer Session 2008, Nida, Lithuania

Lecture 3

Tom Verhoeff

Technische Universiteit Eindhoven
Department of Mathematics & Computer Science
Software Engineering & Technology

Feedback to `T.Verhoeff@TUE.NL`

Topics

- Engineering and Errors
- Verification by Review
- Architecture

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 later a defect is discovered, the higher the costs: grows exponentially in amount of time between injection and discovery.
- Defects decrease the predictability of a project: cost (time) of defect localization and repair are extremely variable.
- Defects concern risks, i.e. uncertainty; product could be defect-free at once, but defects are likely.

Dealing with Defecs

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

Preventing Defects (or instant detection and repair)

- Impossible to do for 100%, but prevention offers the biggest gains
- Every defect not prevented increases the cost
- 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; need 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.
See: NMA Summer Session 2006.

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 (assume, 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 easier and quicker.
- 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

- Costs extra effort and time (mostly initially).
- Consider this to be a small pre-paid insurance fee.
- Cost applies equally to standardization, reviewing, testing, . . .
- Not using these techniques increases risks and unpredictability, and hence 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.
- 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 standard.

- 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).

What is Software Architecture?

- The fundamental organization of a system
- embodied in its components,
- their relationships to each other and
- to the environment, and
- principles guiding its design and evolution.

Why Software Architecture?

- Facilitates parallel construction by a team.
- Improves ability to make work plans.
- Improves verifiability (makes it easier to get it to work):
  - Allows unit testing of separate components.
  - Allows stepwise integration (no "big bang").
- Improves maintainability: most changes affect few components.

Architecture Description: Ingredients

- Stakeholders
- Viewpoints
- Architectural views
- Inconsistencies and conflicts among views
- Rationale

Compare to architectural description of buildings: spaces and doors, water supplies and drains, electricity, heating/cooling, fire safety, . . .

Kruchten's 4 + 1 Views

	Static (Structure)	Dynamic (Behavior)
Abstract	Logical	Process
Concrete	Development concerns code in files	Deployment concerns processors

+ Use case scenarios traced through the architecture

Example of Logical View

How to Design an Architecture

Almost any architecture can be made to work, that is, can be made to provide required *functionality*.

Extra-functional requirements should drive the architectural design: understandability, verifiability, efficiency, maintainability, . . .

Approaches: Top down, bottom up, yo-yo, functional decomposition, data distribution

KISS: Keep It Simple, Stupid

Consider alternatives and compare them: on paper, by experiment.

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
- Example: *Anagrams Architectural Design Document*