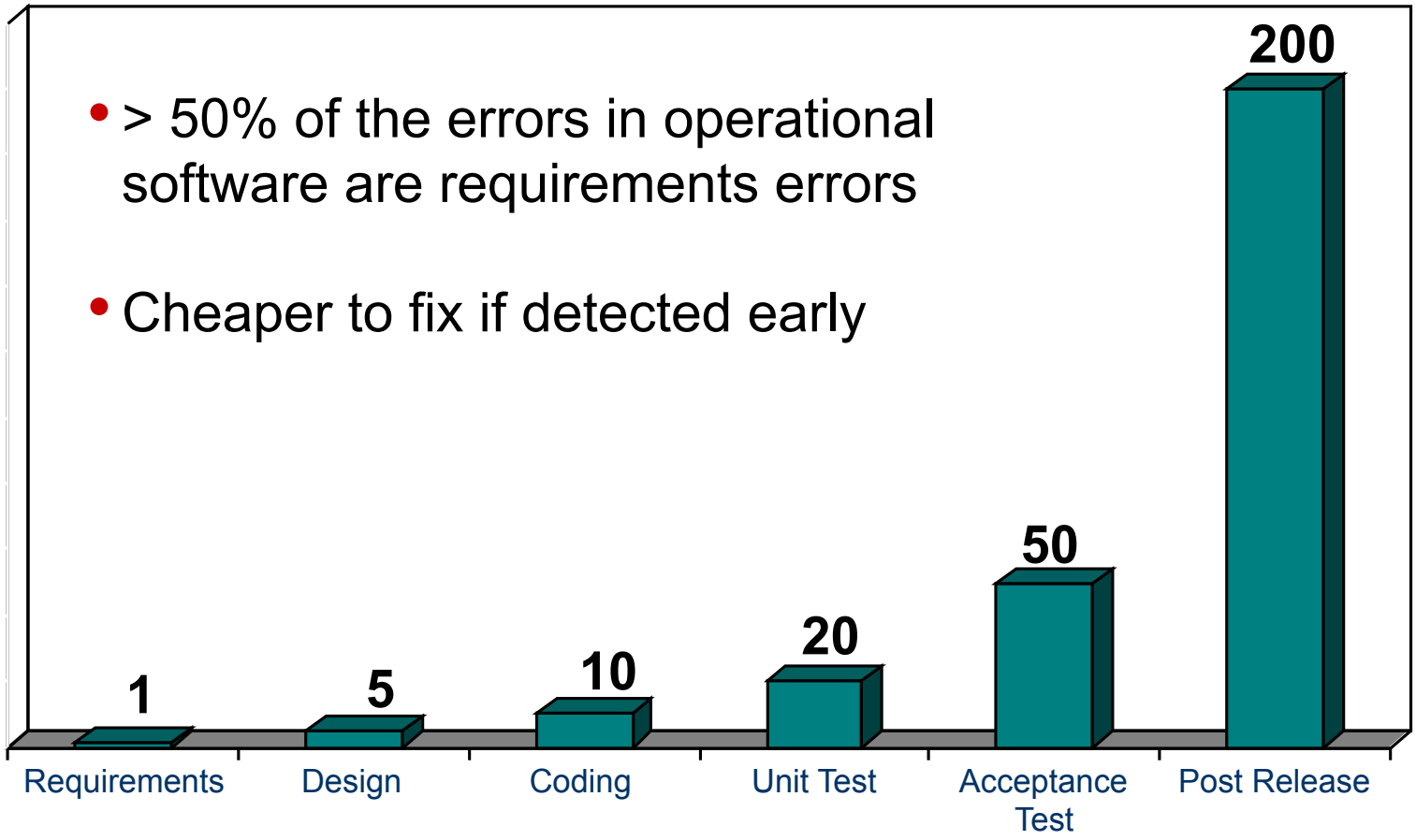
SE463
Software Requirements Specification & Analysis

Validation and Verification

Recall

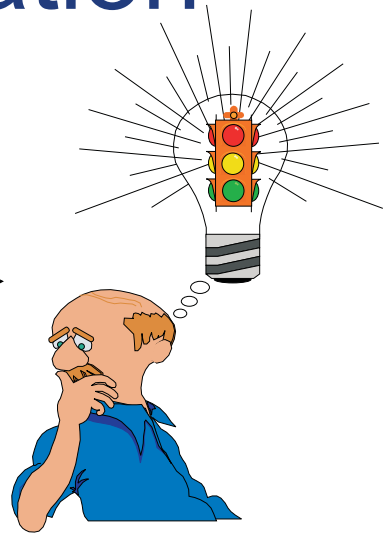
- > 50% of the errors in operational software are requirements errors
- Cheaper to fix if detected early



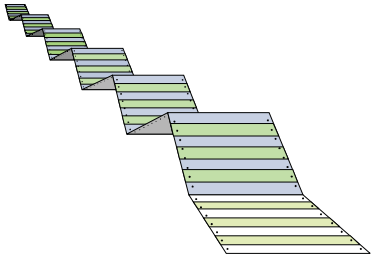
Cost to fix req error once discovered

Validation vs. Verification

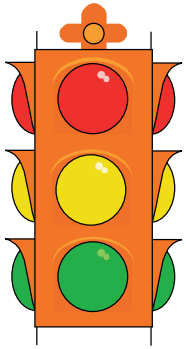
Validation:
Are we building the right product?



Requirements Specification



Verification:
Are we building the product right?



Validation Goals

We want to ensure that

- The specification captures the stakeholders' requirements
- The customer/user is satisfied that the system as specified will fit their needs
- The developers understand the specification

Validation Criteria

- Well-formed
  - unambiguous
  - consistent
  - ranked for importance and stability
  - modifiable (easy to change specification)
  - verifiable
- Valid
  - correct
  - complete

Reviews

- Humans (often semi-outsiders) read and analyze artifacts, look for problems, meet to discuss these problems, and agree on a set of actions to address the identified problems.

- Reviews work.
  - They find more errors than testing does.
  - They find errors faster than testing does.
  - Everyone believes in them

Reviews

There are a number of techniques for reviewing specifications.

- Peer reviews
  - Quota on positive and negative comments
  - Signing Off
- Group Reviews:
  - Walkthroughs
  - Focused Reviews
  - Formal Inspections
  - “Active Reviews”

Walkthroughs

Expert or author presents the specification (scenarios, models, sketchboards, storyboards), and the audience reviews the work from the presentation.

Advantages:

- no prep work, so reviewers may be more likely to attend than if they had to read the document in order to participate
- may be the only way get feedback from busy stakeholders (e.g., customers)
- if the requirements / specifications are presented to a large audience, representing a broad cross section of skills and viewpoints, then there is a hope that major oversights will be caught

Focused Reviews

The reviewing task is decomposed into subtasks, where reviewers have roles and each looks only for specific types of errors (e.g., per use case, or per error type)

Advantages:

- avoids the problem of reviewers not having the time to read the whole document
- each reviewer is more effective because he or she can focus on a handful of error types
- the leader can assign each reviewer to tasks for which he or she is the most skilled

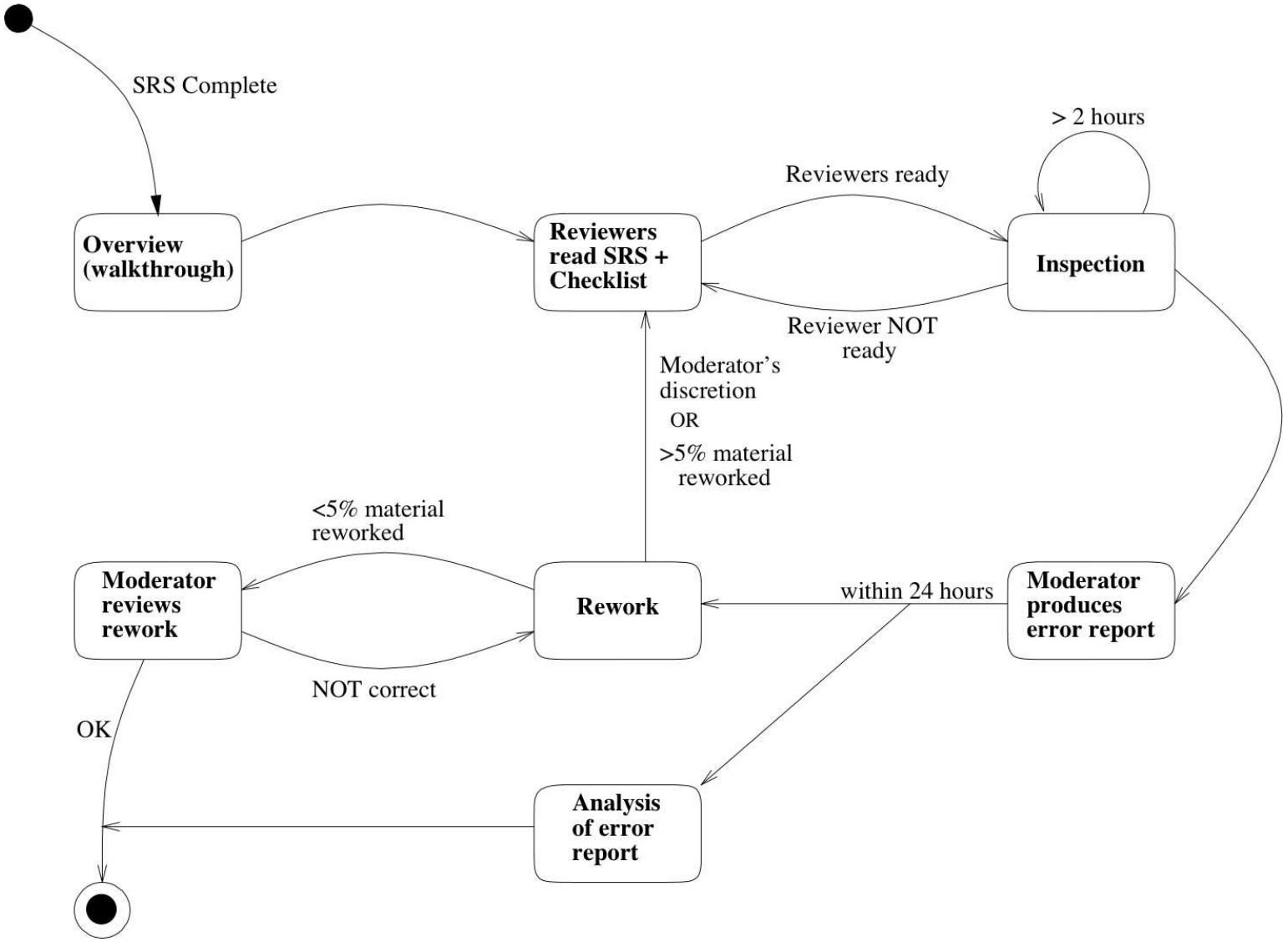
Formal Inspections

Extreme inspection!

A formal inspection is a managed review process

- with rules concerning participants and roles
- strict entry and exit criteria for each step in process.
- primary goal is to improve the quality of the SRS.
- secondary goal is to improve the quality of the development process

Formal Inspections



Formal Inspections

What can an analysis of detected errors tell us?

- It can reveal new types of errors that should be added to the checklists to help with future inspections.
- It can point to projects or use cases that are likely to be problematic, because significantly more errors were reported than usual.
- Evaluation of entry and exit points can help determine whether the project is on schedule.

NASA

NASA requires peer reviews and inspections to be performed on requirements, designs, code, and test plans for projects developing the following software classes:

- **Human Rated Space Software Systems**
- **Non-Human Space Rated Software Systems or Large Scale Aeronautics Vehicles**
- **Mission Support Software or Aeronautic Vehicles, or Major Engineering/Research Facility Software**
- **Basic Science/Engineering Design and Research and Technology Software** (if safety-critical)

It is up to the program/project/facility manager to determine the type of inspections or peer reviews to conduct and on which software products. Software Formal Inspections are recommended for the most critical products and should be performed early, on models, requirements, and preliminary design products, to provide the greatest return on finding and correcting errors and defects before they are propagated into detailed design, code and operations.

NASA, *Software Engineering Requirements*, NPR 7150.2B, 2014

Example Checklist (NASA)

System Requirements Inspection

- Describe proper allocation of functions to software, firmware, hardware, and operations.
- Address the validation of all external usage interfaces.
- Check that all the software system functions are identified and broken into configuration items, and that the boundary between components is well-defined.
- Check that all configuration items within the software system are identified.
- Check that the identified configuration items provide all functions required of them.
- Check that all interfaces between configuration items within the software system are identified.
- Address the correctness of the software system structure.
- Check that all quantifiable requirements and requirement attributes have been specified.
- Address the verifiability of the requirements.
- Check for the traceability of requirements from mission needs (e.g., use cases, etc).
- Check for the traceability of requirements from system safety and reliability analyses (e.g., Preliminary Hazard Analysis (PHA), Fault Tree Analysis (FTA), Failure Mode and Effects Analysis (FMEA), hazard reports, etc.).

NASA, *Software Formal Inspection Standard*, NASA-STD-8739.9, 2013

Example Checklist (NASA)

Software Requirements Inspection

- Check that the specification of each of the following is complete and accurate:
  - Software functions.
  - Input and output parameters.
  - States and modes.
  - Timing and sizing requirements for performance.
  - Interfaces.
  - Use Cases if available.
- Check that specifications are included for error detection and recovery, reliability, maintainability, performance, safety, and accuracy.
- Check that safety requirements are identified as such.
- Check that safety-critical modes and states, and any safety-related constraints, are identified.
- Address the traceability of requirements from higher level documents.
- Check that the requirements provide a sufficient base for the software design.
- Check that the requirements are measurable, consistent, complete, clear, concise, and testable.
- Check that the content of the software requirement specification fulfills the NPR 7150.2 (NPR 7150.2 revision A requirement SWE-109).

NASA, *Software Formal Inspection Standard*, NASA-STD-8739.9, 2013

“Active Review”

Inspection process where reviewers (who are often outsiders) act as users of our specification artifacts.

- Testing team can design tests based on the specification artifacts.
- Technical writers can write the user’s guide or help pages, based on the specification artifacts.

Reviews

Advantages

- More effective than testing (code)
  - Finds causes of errors
  - Reduces errors by factor of 5-10
  - Improves productivity by 14%-25%
  - Reduces V&V costs by 50%-80%
- Simple, doable, only “costs” time and effort
- Inspires authors to improve their work
- Can apply to all sorts of software artifacts
- Improvement claims are backed up by experiment

Disadvantages

- Some find it dull work
- Requires preparation, paperwork
- Discussion can be bogged down in disagreements or details

Testing

Can “run” an executable model and see if its executions match expected behaviour.

Advantages

- Customer can see animation of the product executing
- Checking of low-level details is usually done reliably when done by tools
- Can be done earlier in the development lifecycle than other testing

Disadvantages

- Can be labour intensive and costly
  - hand-holding of tools
  - designing of test cases
- Many notations are not executable
- Testing can be used to show the presence of errors, but not their absence (Dijkstra)

Prototyping

"A software prototype is a partial implementation constructed primarily to enable customers, users, or developers to learn more about a problem or its solution."

Alan Davis

- Presentation Prototypes
  - used for proof of concept; explaining design features; etc.
  - explain, demonstrate and inform – then throw away
- Exploratory Prototypes
  - used to determine problems, elicit needs, clarify goals, compare design options
  - informal, unstructured and thrown away.
- Breadboards or Experimental Prototypes
  - explore technical feasibility; test suitability of a technology
  - typically no user/customer involvement
- Evolutionary (e.g. "operational prototypes", "pilot systems"):
  - development seen as continuous process of adapting the system
  - "prototype" is an early deliverable, to be continually improved.

Throwaway or Evolve?

Throwaway Prototyping

Purpose:

- learn more about the problem or its solution
- discard after desired knowledge is gained

Use:

- early or late

Approach:

- horizontal - build only one layer (e.g. UI)
- "quick and dirty"

Advantages:

- learning medium for better convergence
- early feedback → less cost
- successful even if it fails!

Disadvantages:

- wasted effort if reqts change rapidly
- often replaces proper documentation of the specification
- may set customers' expectations too high
- can get developed into final product

Evolutionary Prototyping

Purpose:

- learn more about the solution
- reduce risk by building parts early

Use:

- incremental; evolutionary

Approach:

- vertical – partial implementation of all layers
- designed to be extended and adapted

Advantages:

- requirements are not frozen
- return to last version if problem encountered

Disadvantages:

- early architectural choice may be poor
- can end up with complex, unstructured system that is difficult to maintain
- optimal solutions not guaranteed
- lacks control and direction

Safety Analysis

Safety analysis looks for safety hazards in the system: safety-related failures, their causes, their consequences.

Software Faults

- Data sampling rate
- Data collisions
- Illegal commands
- Commands out of sequence
- Time delays, deadlines
- Multiple events
- Unsafe modes

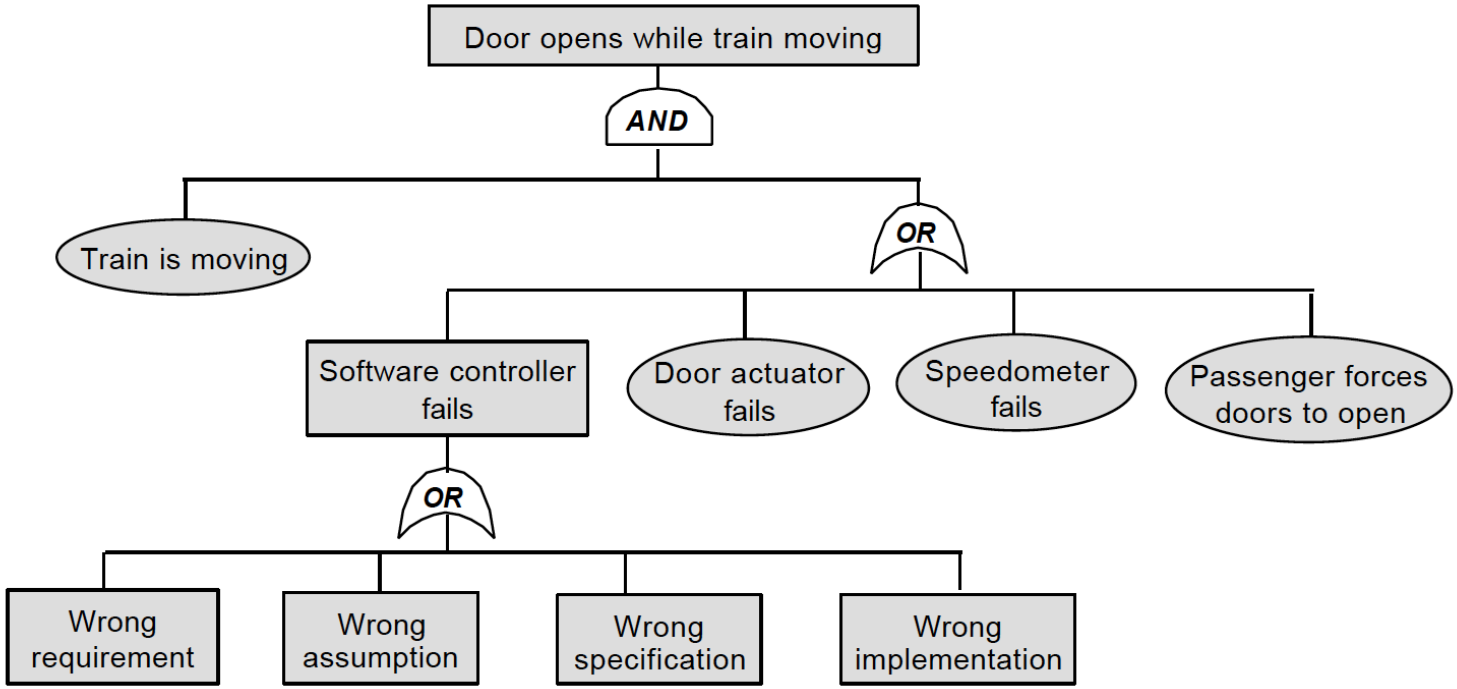
Failures in Environment

- Broken sensors
- Memory overwritten
- Missing parameters
- Parameters out of range
- Bad input
- Power fluctuations
- Gamma radiation

NASA Software Safety Guidebook

Software Fault Tree Analysis

Software Fault Tree Analysis (SFTA) is a top-down backward-analysis approach to safety analysis that starts with an undesired software event and determines all the ways it can happen.



van Lamsveerde, *Requirements Engineering - From System Goals to UML Models to Software Specifications*, Wiley, 2007

Independent V&V

V&V performed by a separate contractor

- Independent V&V fulfills the need for an independent technical opinion.
- Cost between 5% and 15% of development costs
- Studies show up to fivefold return on investment:
  - Errors found earlier, cheaper to fix, cheaper to re-test
  - Clearer specifications
  - Developer more likely to use best practices

Summary

- Validation: check you are solving the right problem
  - Walkthroughs
  - Inspections
  - Active reviews
  - Prototypes
- Verification: check that your development steps are sound
  - Testing
  - Safety analysis
  - Consistency checking