Validation

Notes by mainly Jo Anne Atlee,
with modifications by Daniel Berry
dberry@uwaterloo.ca

Fall 2004

Intro to Verification and Validation[a]

Validation — Evaluating a software artifact (*e.g.*, reqs spec) wrt customer reqs.

$\rightarrow$ *"Are we building the right system?"*
i.e., is the spec what the customer really wants?

$$\boxed{\text{Spec, Domain} \models \text{Req}}$$

Verification — Evaluating a software artifact wrt existing artifacts.

$\rightarrow$ *"Are we building the system right?"*
e.g., does the design implement the spec?

$$\boxed{\begin{array}{c}\text{Design} \models \text{Spec} \\ \text{Code} \models \text{Design} \\ \text{TestCases} \models \text{Spec}\end{array}}$$

[a] You will encounter these topics again in SE-2 and SE-3.

Therefore, our focus is on Validation

Validation Criteria

... which are what you expect:

- correctness
- unambiguousness
- completeness
- consistency
- accuracy
- traceability
  - $\rightarrow$ origin of each specified behaviour is clear
  - $\rightarrow$ spec'd behaviour is labelled so other artifacts can reference individual requirements

Requirements Validation

Requirements validation involves checking the correctness, completeness, and accuracy of the requirements specification

Requirements validation is the last stage of requirements engineering, but ...

don’t wait until the end to start validation. You might discover serious problems that cause a complete rewrite of the specifications. You need to validate continually with the client, at every step along the way.

The difficulty with requirements validation is that there are few documents which can be used as the basis for validation. A design or program can be verified against the specification, to see if it implements the specified behaviour. When validating a specification, we're validating it against the stakeholders' requirements. Some of these may be documented. If they are documented, they are probably expressed in natural language, which probably means they are open to multiple interpretations. All of this means that thoroughly validating a document is a time-intensive and error-prone process.

Validation Techniques

- Testing
- Reviews

Validation Techniques[a]

1. "Testing" (*i.e.*, machine processing artifacts)

→ Usually, we mean testing of code, but we can mean also testing an executable specification.

→ Could also mean just automatic tool checking of artifacts, *e.g.*, type checking a Z specification, or using a model checker.

→ Testing comes *after* delivery of the artifact and is *not* performed by the developers.

[a] Many taxonomies are possible.

- Advantages:
  - Low-level details checking is usually more reliable when done by tools.
- Disadvantages:
  - Very labour intensive (hand holding of tools, designing of test cases) and costly.
  - Not clear when to stop. Law of diminishing returns definitely a factor in testing.
  - Many specification notations are not executable or even (usefully) checkable.
  - Remember that testing can be used to show the presence of errors but not their absence.

Testing — simulate executable specifications

If you have an executable specification, you can simulate it on various input and see what the specified output is. Simulation has all of the benefits and pitfalls of testing, but it can be done earlier in the development lifecycle than most other testing.

Does not escape the main pitfall of testing, namely that it can be used to show the presence of errors but not their absence.

Better yet, you can ask the customer or future users to run some simulations and see if the behaviour they see is what they expected.

Validation Techniques

2. <u>Manual reviews/inspections</u>

- Let humans (often semi-outsiders) look at artifacts carefully. Often, they will have a good idea of likely problem areas both inside and outside problem domain.

  $\rightarrow$ Need both domain experts and domain-ignorant developers.

- FACT: Reviews work, period.

  $\rightarrow$ They find more errors than testing does.

  $\rightarrow$ They find errors faster than testing does.

  $\rightarrow$ Everyone believes in them, even Microsoft.

- Advantages:
  - Can review all kinds of software artifacts, not just code, *e.g.*, specs, test suites, design docs
  - Errors are caught much sooner than testing, when they are much cheaper to fix!
  - Can serve to bring the entire development team together into the “big picture”, to educate newcomers.
- Disadvantages:
  - $\rightarrow$ Some find it dull work.
  - $\rightarrow$ Often requires preparation, paperwork.

Requirements Reviews

Requirements reviews are the most widely used technique of requirements validation. They involve a group of people who read and analyze the requirements, look for problems, meet to discuss these problems, and agree on a set of actions to address the identified problems.

There are a number of specific techniques of requirements reviews.

- Reading
- Signing Off
- Walkthroughs
- Formal Inspections
- Focussed Inspections
- Active Reviews

Reading the Document

A review can be as simple as reading the document, looking for errors. All technical reviews, despite their differences, are based on the idea that developers are blind to errors in their own work, and that it is beneficial to have someone else look at their work.

The last person you want proofreading a document is the author, because he or she reads what he or she *thinks* the document says, not what it *actually* says!

The problem is how to get reviewers to do the work. Reviewers are rarely professional reviewers. They are usually software developers that have their own work they need to do. They have their own deadlines and will give their own work higher priority.

Signing Off

Asking reviewers to sign off on documents makes them partly responsible if errors are subsequently found in the document. By signing off on a document, a reviewer is saying that he or she has reviewed the document and approves it. This approach to reviewing often encourages reviewers to be more thorough. If a reviewer’s name is going on the document, he or she will be more inclined to make sure that it is correct.

Types of Group Reviews

- Walkthrough
  - → Informal, often high-level overview.
  - → Often led by author/expert to educate others on his/her work.
  - → Goal may be knowledge transfer or finding errors or both.
- Formal inspection [Fagan 76]
  - → Structured inspection of, say, code.
  - → Usually, a *very* detailed examination of an artifact.
  - → Participants have defined roles; preparation required; paperwork generated.

Walkthroughs

In a walkthrough, an expert or the author presents the specification, and the other participants ask questions and give comments.

Walkthroughs are informal meetings, in which an expert or the author presents the specification and the audience reviews the work from the presentation. Participants may have different levels of understanding going into a walkthrough, so walkthroughs can also be tutorials.

An advantage of walkthroughs is that they don’t make many demands on the participants, so reviewers may be more likely to attend than if they had to read the document in order to participate.

Walkthroughs may be used more often in reviews of requirements documents than in reviews of other software documents. Reviews of requirements documents involve a large number of people, because there are usually a large number of stakeholders to consult, and it may prove impossible to get everyone prepared for a more formal review. In such cases, a walkthrough may be the only reasonable way to ensure that the the stakeholders have actually looked at the material.

Another perceived advantage is that if the requirements are presented to a large audience, preferably one that represents a broad cross section of skills and viewpoints, then there is a hope that there are no major oversights in the requirements. In other words, mulitple heads are better than one, and redundancy helps.

Formal Inspections

A formal inspection has a managed review process, with rules concerning participants and roles, and with strict entry and exit criteria for each step in process.

The idea behind formal inspections is to improve the quality of the document. If the purpose of the walkthrough is to gain some assurance that there are no major oversights in the requirements document, then the purpose of the formal inspection is to strive for a zero-defect document.

Formal inspections are characterized by rules on who should participate, how many reviewers should participate and what roles they should play.

There should be from 3 to 5 reviewers: author, moderator$\neq$author, and other reviewers

- The author serves as the presenter of the document.
- The moderator initiates the inspection, convenes the meeting, assigns roles, controls the meeting, decides whether to do another inspection, and prepares as the other reviewers.
- The other reviewers prepare for inspection by reading the document and documenting errors. These reviewers will often have checklists of common errors, perhaps different for each reviewer

Inspection Meeting

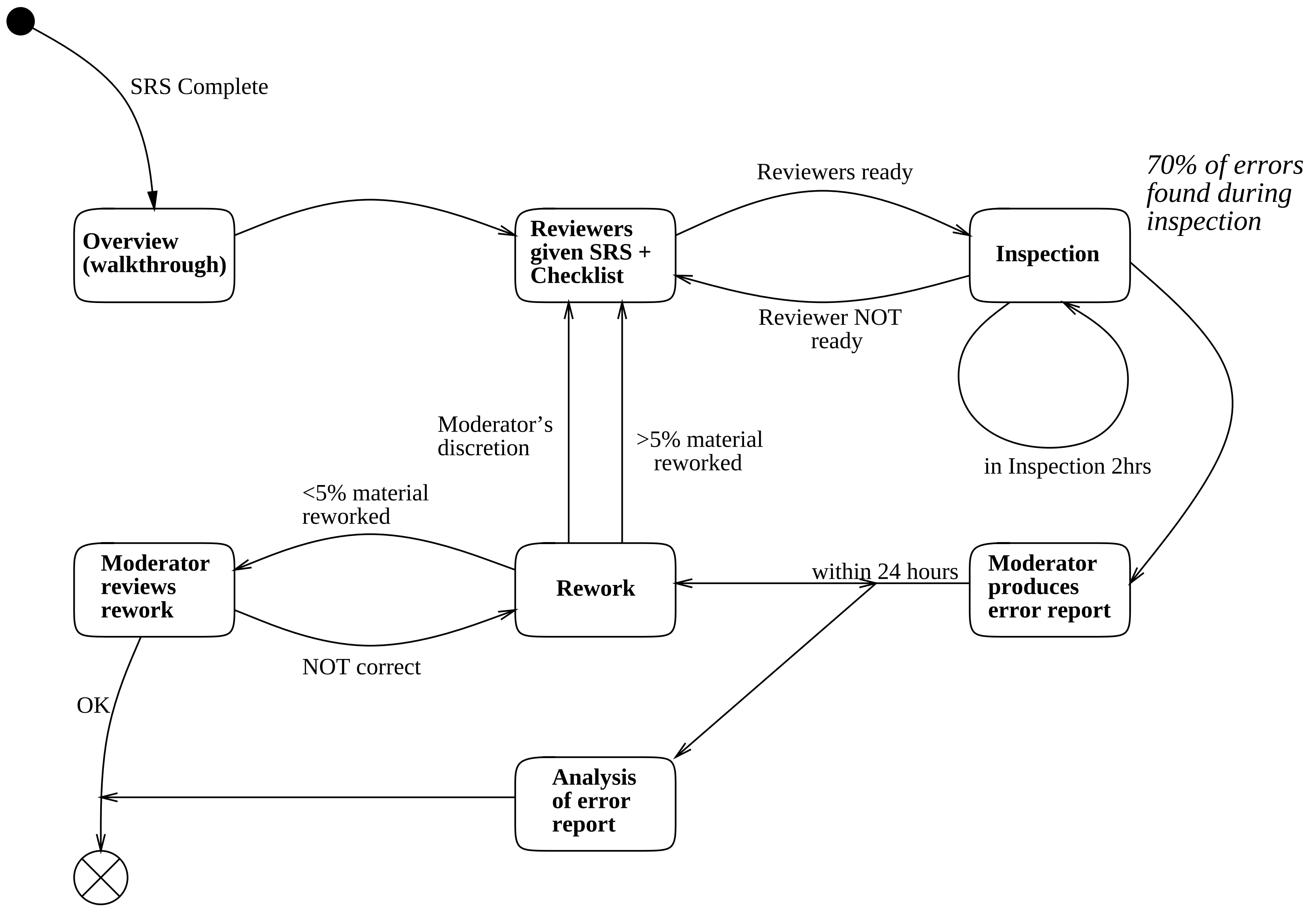
Prior to the meeting, there is a walkthrough to familiarize the reviewers with the document to be inspected.

The reviewers receive copies of the document, and each prepares for the inspection meeting by reviewing the document privately to find as many problems as possible, possibly according his or her checklist.

The focus of the inspection meeting is on finding problems, and not on fixing them. No time is wasted to fix problems; indeed, a fix may be invalidated by a problem or fix found later. Fixing is left to the author after the inspection meeting.

The moderator’s main job at the inspection meeting is to keep the focus on finding problems and to cut off any digression to solution finding.

Fagan Inspections



My own opinion is that if *any* of the material is reworked, not only if more than 5% of the material is reworked, you should do yet another inspection.

It is too easy to introduce new problems when fixing old problems and these may need to be found by inspection.

The inspection meeting can be thought of as the first part of a brainstorm to locate problems in the inspected document. It is only the first part, because the pruning takes place privately as the author reworks the document, learning in the process which of the reported problems are indeed problems.

Inspection meetings are postponed also after 2 hours. It has been observed that reviewers’ error detection rates go down after 2 hours, and it is better to wait and continue only when the reviewers are fresh.

Inspection is complete only when rework is complete.

Error data are collected, reported, and analyzed.

Very Important: the author’s boss is not allowed to sit in on review or to see the the data. Inspections are not to be used for employee evaluation; they are to be used only for collecting error data so that the software can be fixed and future inspections can be improved.

One of the motivations behind formal inspections is to give management a way of measuring and managing quality assurance. What can an analysis of detected errors tell us?

- It can reveal new types of errors that should be added to the checklist to help with future inspections.
- It can point to projects that are likely to be problematic, because significantly more errors were reported than usual.

Focussed Inspections

In a focussed inspection, the reviewers have roles and each looks only for specific types of errors.

Focussed inspections help avoid the problem of reviewers not having the time to read the whole document. Also, the leader can assign each reviewer to tasks for which he or she is most skilled. A reviewer who is an expert on the requirements can look for missing and wrong requirements. A UML expert can look for modelling errors and not be an expert about the systems requirements. Those who are skilled at and enjoy finding inconsistencies, and who may not be experts on anything in particular, can be set loose looking for inconsistencies. (Later, we learn about the smart ignoramus!)

Active review — ask reviewer to use the specification

In this case, the author poses questions for the reviewer to answer that can be answered only by reading the document. Not only does this force the reviewer to do the work, but it also exercises the document in ways that it will be exercised in practice by software users.

One example of such a question is, “Find requirements that justify every specified method.”

Another example is to give each reviewer a different set of scenarios and ask each reviewer to walk through each scenario with the specification, to make sure that the specification handles the system’s role in each scenario

- Active review [Parnas]

  → Inspection process where reviewers (who are often outsiders/ignoramuses) act as users of the artifact.

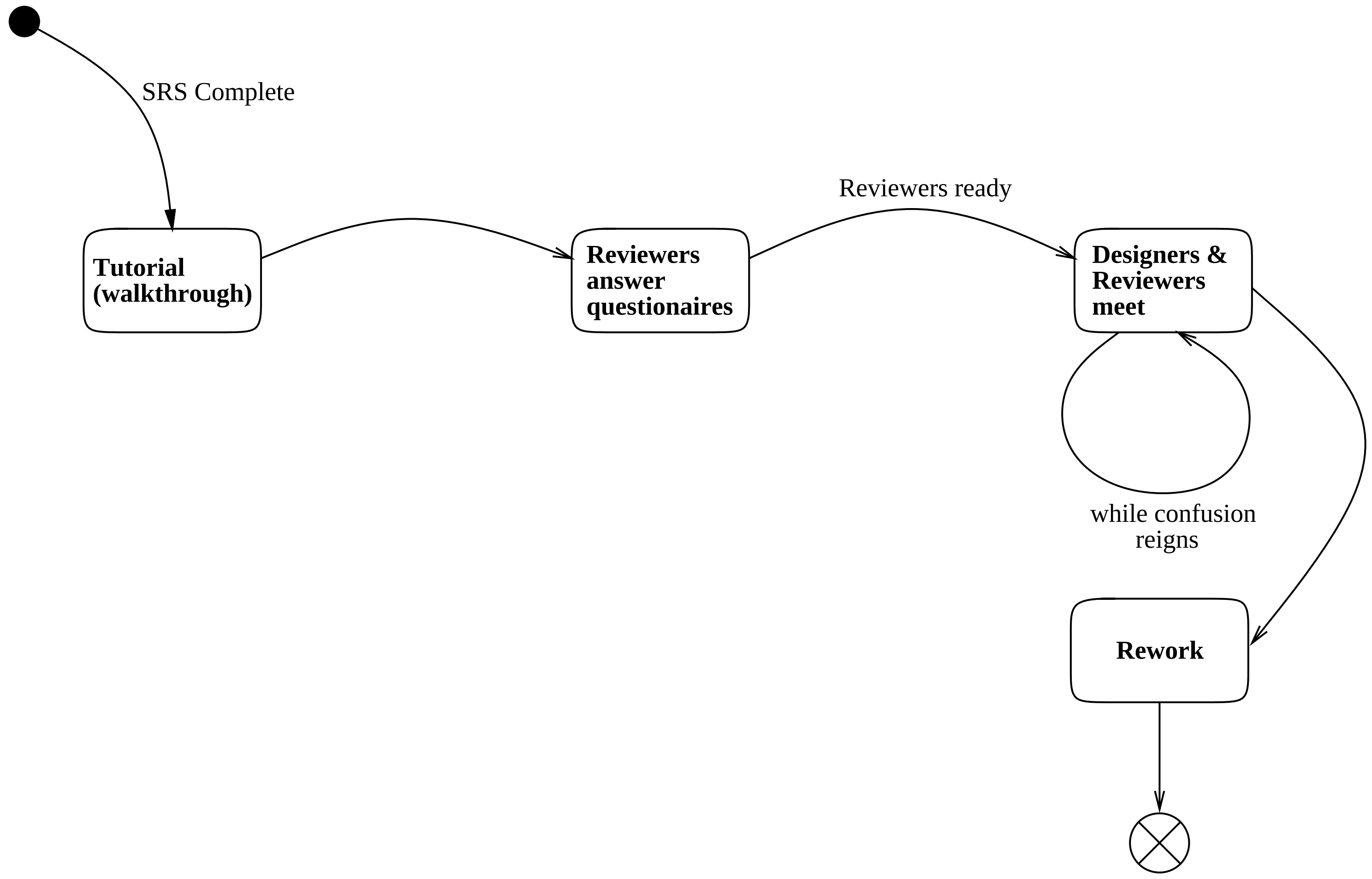
  → Authors pose questions that require reviewers to use artifact to answer.

  → Like a kind of pre-alpha release with specific demands on users.

  Example: *For each of the access functions, the reviewer should answer the following questions:*

  1. *Which assumptions tell you that this function can be implemented as described?*
  2. *Under what conditions may this function be applied? Which assumptions described those conditions?*
  3. *Is the behaviour of this function (its effects on other functions) described in the assumptions?*

Active Reviews



Your TA's evaluation of your SRS is a review. How do they review your document?

- The evaluation is basically a reading type of review, but it has some aspect of a sign-off review. In some sense, the TA's job *is* to review your work. Thus, they are professional reviewers, and they are more likely to do a thorough job than a reviewer in practice
- They have a marking scheme, which acts as a form of checklist of errors to look for. The marking scheme lists key functionality for them to look for. It lists also common properties to evaluate, such as level of detail, clarity, model layout, etc.

- They have a list of common UML and SDL problems.
- They have key scenarios that they use to debug your models.

Advantages of Code Inspections

- Considered to be more effective than testing for finding bugs:
  - → Testing finds the *symptoms* of errors.
  - → Inspections find the *causes* of errors.
- Authors write their code expecting *others* to be able read and understand it; often improves programming habits! Also, author learns from inspections what makes programs understandable.
- Author often develops “blind spots” about his/her code:
  - → Fresh eyes may spot errors/flaws more easily.
- Having to explain something is an excellent way to learn it! (Every teacher knows this!)
- Through them, a company can enforce coding standards.

Potential Problems?

- personality problems:
  - person with good ideas may not be able to express them well
  - person with bad ideas might dominate
  - some people dislike disagreements, others love arguing for its own sake
  - "holy wars": sometimes people have fundamentally irreconcilable points of view
  - "semi-colon wars": easy to get lost in trivial matters
- office politics:
  - all comments get logged formally; can get back at people you don't like[a]
- it's draining; loses effectiveness after a couple of hours

[a]BUT: recall rule about the author's boss not being present

BNR Code Inspection Example

BNR[a] produced about 20 MLOC of source code over 10 years. Their DMS digital switching software is about 10 MLOC.

- They inspected 2.5 MLOC, 8 releases over 2 years.
- They found 0.8 – 1.0 errors per person-hour by inspection, which is 2 to 4 times more effective than testing.
- They found about 37 errors per 1000 LOC. (Other studies found 50–95 errors per 1000 LOC.)

  Type of errors:

  – 50% incorrect statement

  – 30% missing statement

  – 20% extra statement

[a]Bell Northern Research is now known by the parent company's name NorTel aka Northern Telecom. They are a huge telecommunications company based in Ottawa.

- An error diagnosed in released software takes $\approx$ 33 person-hours to diagnose and repair.
- Their coders typically produce 3 to 5 KLOC of finished, documented code per person-year. This is difficult software to get correct (large, real-time system)!

Why Do Inpsections Work?

[Discuss]

- It’s a popular, time-tested technique.

  → Microsoft believes in it strongly!

- Simple, doable, costs “only” time and effort
- some *very* impressive experiences, and (unlike many other claimed SW improvements) they have high credibility
- goal is detection and improvement NOT witch hunting, programmer evaluation, scorekeeping, management spot checks

  → “It’s OK to be wrong.”

- when you know your work will be reviewed, you tend to be more careful and follow B&D practices (commenting, *etc.*)

Why Do Inpsections Work?

- side effects:
  - encourages group “buy in”, everyone is now familiar with the code, wants to make it work
  - encourages handing down of knowledge from old hands to newbies
  - encourages adherence to coding standards
    (so you have common vocabulary/expectations)
- (ideally) reduces time needed for testing, with less overall effort

Finally ...

- Amount of structure and formality in process varies widely:
  - go through code line-by-line
  - everyone read beforehand, report only problem spots
  - asynchronous reviews
- Web-based techniques are currently hot:
  - put up code on web page (usually within company network!)
  - reviewers can be geographically distributed, different timezones
  - review asynchronously or via groupware
- other similar ideas:
  - Web day
  - Demo day (for peers or customer)