

Chapter 6



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 6: Requirements Quality Assurance

DTU course 02264

Abstract

- In this chapter we first revisit some famous software failures and discuss how they relate to Requirements Engineering.
- Next we go through the most common problems and pitfalls when working with requirements. According to their main trigger, they are classified into people, process, individual requirement, and requirement set issues.
- We introduce formal inspections and error density estimation as two „V&V“ techniques and conclude with remarks on the nature of specifications.

Contents

1. Notions of Quality
2. Design Inspection
3. People Issues
4. Process Issues
5. Individual Requirement Issues
6. Requirement Sets Issues
7. Error Density and Estimation



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 6.1:

Notions of Quality

DTU course 02264

Some problems with requirements

My son's Christmas wishlist

A fast car

Shavn the Sheep

The bicycle from the shop next door

The yellow fish from the aquarium

The toy that Miriam brought to
Kindergarden yesterday

What is a good requirement?

- **Any requirements must satisfy the following quality fundamental criteria. It must be**
 - Correct (technically and legally possible)
 - Complete (express a whole idea or statement)
 - Clear (unambiguous and not confusing)
 - Consistent (not in conflict with other requirements)
 - Verifiable (it can be determined that the system meets the requirement)
 - Traceable (uniquely identified and tracked)
 - Feasible (can be accomplished within cost and schedule)
 - Modular (can be changed without excessive impact)
 - Design-independent (do not pose specific solutions on design)
 - Modifiable (can be changed to adapt to new knowledge)
 - Usable (during Operation/Maintenance)
- **Many of these properties have been defined a long time ago (e.g., in the IEEE Standard 830), but they are still not common-place.**

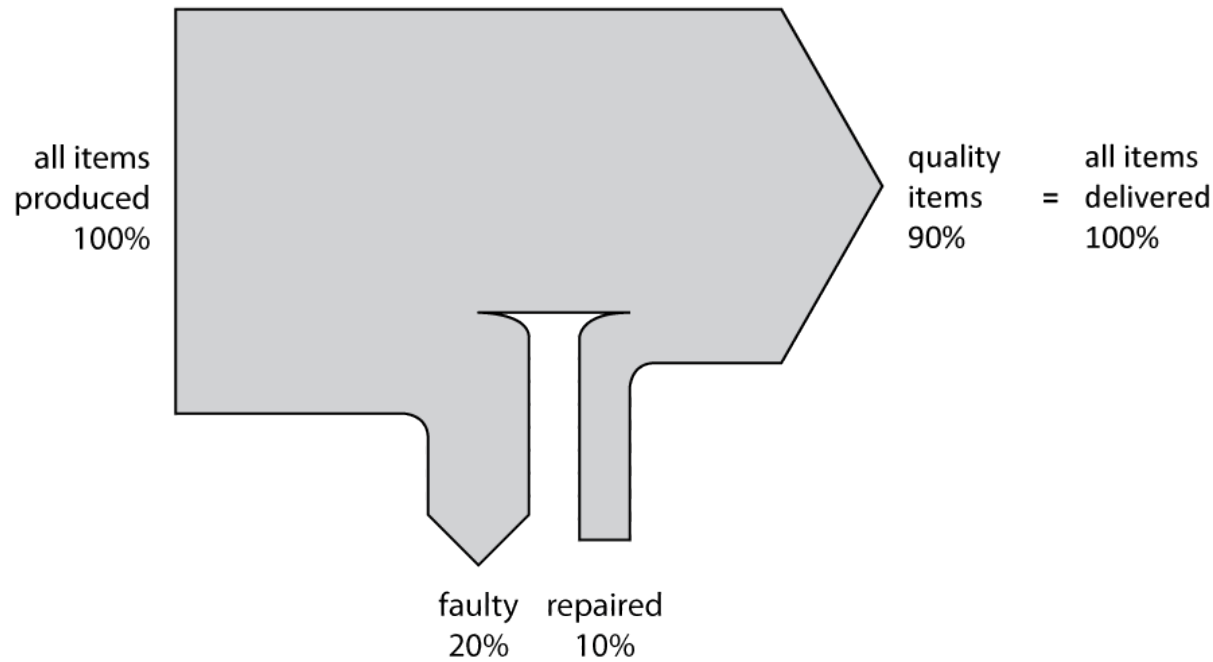
Requirements Quality vs. Quality Requirements

- **Quality requirements refer to those requirements that are concerned with the quality of the system under design.**
- **Requirements quality refers to the quality of the models, documents, and other artifacts created as part of the RE process.**
- **The quality of the system under design is, obviously, affected by the quality of the RE, in particular the quality of the requirements as such.**
 - Without high quality requirements, it is impossible to create a high quality system.
 - However, the opposite is not true, unfortunately: high quality requirements do not guarantee a high quality system built based on them.
- **Either way, high quality of requirements is a desirable goal.**

Constructive vs. Destructive QA

- Traditionally, quality is seen as the absence of faults in delivered products, so that high quality may be assured in two different ways.
 - By removing faulty parts after production (destructive QA); or
 - by creating parts with high quality to begin with (constructive QA).
- In that sense, quality can be “tested into” a manufacturing product.

This is clearly a wasteful process: either, rejected items are destroyed, or they have to be repaired. In the case of software, they have to be programmed from scratch again, or debugged.



Constructive vs. Destructive QA

- **In manufacturing, however, the wastefulness of destructive QA is mitigated by large series of identical products.**
 - In software development, all items and their requirements are inherently different – otherwise, we'd just copy them.
- **In some sense, destructive QA can be adopted for software, too.**
 - For instance, the Space Shuttle software was created twice, and was running concurrently.
 - Explorative prototyping can be seen as an attempt to emulate destructive QA while avoiding some of the cost.
 - Highly iterative processes also exhibit some characteristics of this approach, but the only benefit of repetition is in learning of the underlying (mental) skill of programming,

An Alternative View on Quality

- **Since the 2000's, another view of quality has gained popularity, together with light weight ("agile") development methods, favoring small cycles & process improvement over (*"Abandon perfection for execution!"*).**
 - This approach is often justified by reference to the manufacturing world, in particular, the automotive industry for their appreciable levels of quality, compared to the number of units produced.
 - This comparison is deeply flawed since the error reduction strategies in the manufacturing industry (and others) apply to production rather than development.
 - The development process in the automotive industry is highly restrictive, and today uses almost entirely the model-based paradigm to software development.
- **This is often believed to deliver more value quicker, at better quality.**
 - For a very small class of systems/projects, short-term improvements can be realized, but after some time, the competitive edge disappears.
 - *"I have seen many such projects, and they all had their good reasons to go agile. And it worked, for them! After a year or so, however, they hit a stone wall."*
Francis Bordeleau, Ericsson, MODELS Keynote 2014



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 6.2:

Design Inspection

DTU course 02264

Fagan-Style Inspections

- **An Inspection is a structured procedure to find faults in artifacts.**
 - Inspections are mostly used for code and software architecture, but they may be used in all areas of Software Engineering.
 - In contrast to Reviews (which are much less formal and structured), there are very strict and detailed instructions for inspections (e.g. concerning roles, process, constraints).
 - Inspections may be used at any time during the development and for any kind of artifact.
- **“Proper” Inspections...**
 - use goal-oriented checklists,
 - define roles for members of the inspection team, and
 - keep statistics on discovered faults and duration of inspection.
- **Inspections target at substantial faults (“major defects”).**
 - A major defect is one that can cause significantly increased cost if it is not found now.
 - Minor defects like typos, deviations from conventions etc. can often be found in cheaper and faster ways (i.e. spell checkers, style checkers/auto formatters).

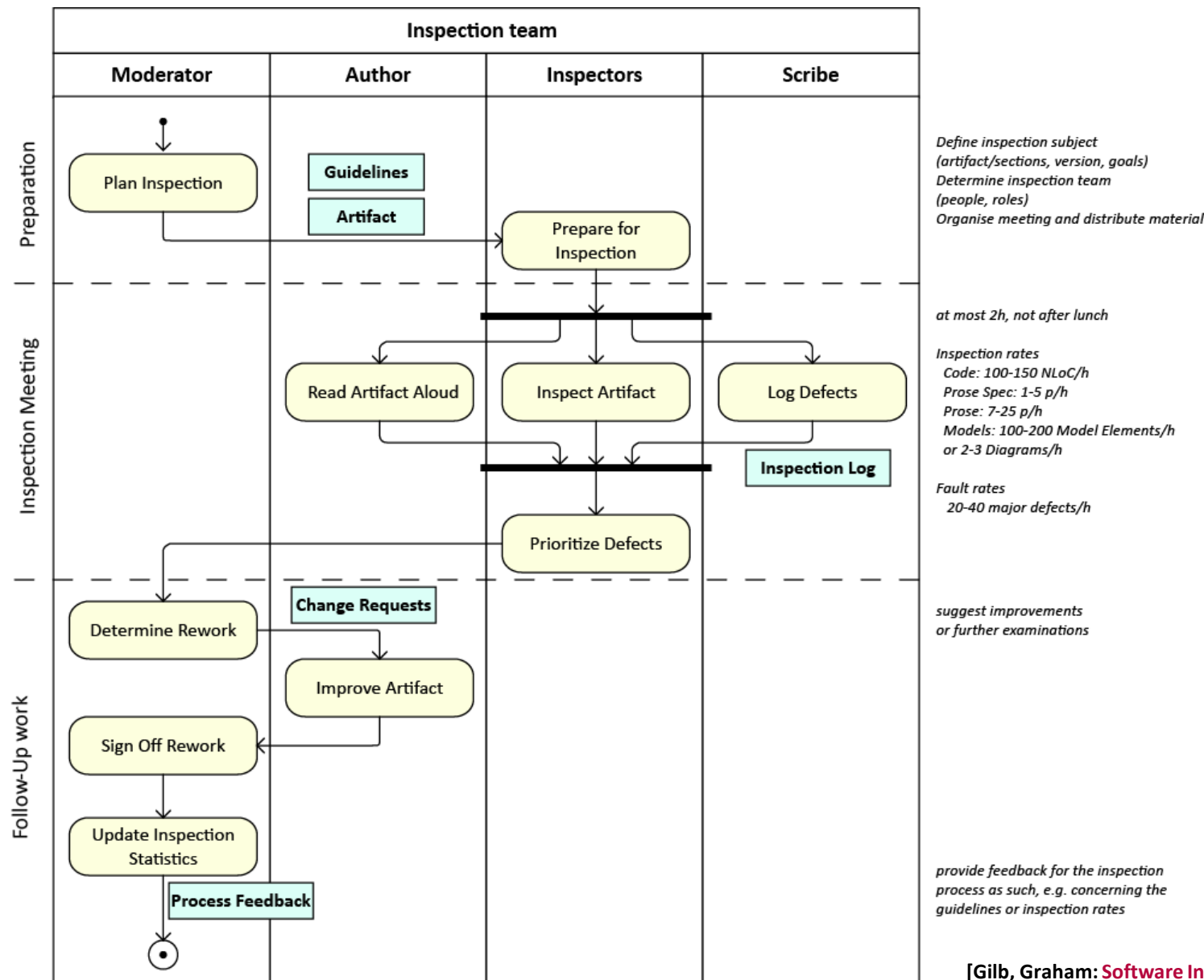
Inspections are effective and efficient

- **More than 60% of all defects can be found by informal inspections.**
 - Fagan, 1960
- **More than 90% of al defects can be found by formal inspections.**
 - Mills et al., 1987
- **Inspections are more effec tive and cheaper than tests.**
 - Selby & Basili, 1987
 - Gilb & Graham, 1993
- **Inspections have been identified as the root cause of substantial increases in productivity (10...30%).**
 - 14% (ATT Bell Labs);
 - 25% (Aetna Insurance Comp.);
 - 30% (Gilb: Sw. Metrics)
 - op cit. Humphreys: Managing the Software Process, p.186

Manual Inspection vs. Automated Test

- **Inspections can be applied to all kinds of artifacts, automated tests require a running system.**
 - Inspections can be applied to all artifacts and any time during development, in particular in the early stages, when defect removal is most valuable.
- **Tests can only show the presence of errors, inspections usually provide improvement suggestions, too.**
 - If a test finds an issue, something very similar to an inspection has to be conducted to define and resolve the issue.
- **Many tests can be automated, so that it is very cheap to repeat them. Repeating an inspection is as expensive as the first round.**
 - If some QA activity is to be repeated many times, automated tests are cheaper. if it is to be repeated only a few times, inspection is cheaper.

Inspection Process



Inspection Roles

- **There are four roles in the inspection process:**
 - Author/Reader: a person presenting the artifact and reading it to the inspection team.
 - Moderator: organizes the process, guides the discussion,
 - Scribe: helps the moderator by taking care of the writing.
 - Inspector: a person assessing the artifact, taking notes in advance.
- **A natural person may take on several different roles.**
 - Only the author is an exception: s/he may only act as an inspector.
- **Only for large/controversial inspections is a scribe necessary.**

Moderator Role

■ Preparation

- Prepare and organize the review, select reviewers, set deadlines, distribute artifacts, guidelines, and forms.
- Compiles the inspector comments before the inspection meeting, fills in review entry checklist.

■ Inspection Meeting

- During session moderator leads the discussion.
- If remarks are made up during inspection or if inspectors remarks are modified, he (or the scribe) notes them down and this one is referred to in the remarks section.

■ Follow-Up work

- The moderator staples sheets together in right order, files and distributes them to the participants. He enters the essential quantitative data into a spreadsheet template.
- He keeps track of the rework assignments and signs off the whole process in the end.

- © 2010, Prof. Dr. H. Störrle

[illegible]

Selection of Inspection Targets

- **Inspections are cost-effective, but they are still effective. In order to maximize the benefit from an inspection, the inspection artifact should be selected carefully.**
- **Select an artifact, that is**
 - Central, i.e. one where mistakes and omissions will have a great impact;
 - Improvable, i.e. one where you expect (or hope for) improvements; and
 - Critical, i.e. one that realizes requirements of great importance.
- **Prepare the artifact such that**
 - the artifact is accepted for inspection (rejection is very embarrassing);
 - there are no trivial complaints (typos, formatting); and
 - The inspectors understand the what, how, and why of the inspection.
- **Handing in sloppy, trivial, risk-free, or irrelevant artifacts is a bad idea.**
 - Make sure your inspectors understand what you want as developers want to get out of the inspection and why.

Inspection Caveats

- **The artifact is being inspected, not the author!**
 - Be tough on faults, but gentle on people.
- **Faults and remedies are not just discussed but written down, followed up on, and eventually signed off.**
- **All team members are informed of all steps until the end.**
 - They are responsible for the outcome, all of them, collectively.
- **It is essential that the inspectors prepare for the inspection.**
 - Read and understand the material (the artifact) at hand.
 - Use guidelines and/or checklists to direct your attention, but don't let them keep you from using your common sense.
 - The moderator compiles inspection remarks in advance to find “hot spots” and guide discussion.
- **Don't start the inspection unless everybody is prepared.**



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 6.3:

People Issues

DTU course 02264



- **Plating water taps with gold does not alter function in any perceivable way, but does add to the cost considerably.**
 - From an engineering point of view, this is to be avoided as it is not cost-effective.
 - From a marketing point of view, it may be just what is needed, though.
- **The Volere-Approach tries to achieve this using two separate dimensions for the customer satisfaction and dissatisfaction.**
 - This is based on results from psychology (Motivator/Hygiene-Theory) that seemingly opposites are often at the ends of two different scales altogether.
 - Classical examples include happiness/unhappiness in relationships or jobs.
 - For instance, great mutual love will cause happiness. Incompatibility of living styles (responsibilities, fidelity, leisure activities, ...) will cause unhappiness.
 - Experiencing happiness and unhappiness at the same time is considered a schizophrenic state.

Avoiding Gold-Plating

- Thus, prioritizing requirements purely on the basis of satisfaction scores is likely to suffer from gold plating.
- There is no doubt that class A requirements ought to be implemented, and that class D requirements may be deferred: they are gold plating requirements.
- But what about classes B and C?
- Class B requirements are so called bread-and-butter-requirements: features that must be in place, but do not add to the attractiveness.

		customer dissatisfaction	
		high	low
customer satisfaction	high	A	C
	low	B	D

So, the overall priority would be

- A-B-C-D for selling the project
- B-A-C-D for reduced overall cost

where, typically, C and D never get implemented.

Developer Gold-Plating

- **Naturally, many developers are more interested in the technology than in the solution.**
 - They may want to explore their language/framework/IDE rather than the client's problem space.
- **Therefore, solutions may pay undue attention to technical detail rather than the big picture or customer value.**
- **Also, complex technical solutions are preferred over minor changes in the business requirements and procedures making these solutions obsolete.**



- Developers and technical managers often believe in solutions that resolve all issues magically.
- For instance, they might say something like this.
 - *“Using UML will improve our modeling”.*
 - *“Using XP will speed up our development”.*
 - *„If only we get the requirements right, the project will be successful.“*
 - *„By using Java, we have eliminated all portability issues.“*
 - *„Making Eclipse our standard IDE will double our productivity“.*
- But as we all (should) know ***„There is no Silver Bullet“*** (F.P. Brooks in ***„The mythical man-month“***).
 - Believing in it anyway is a sign of inadequate professional standards and lack of knowledge.



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 6.4: Process Issues

DTU course 02264

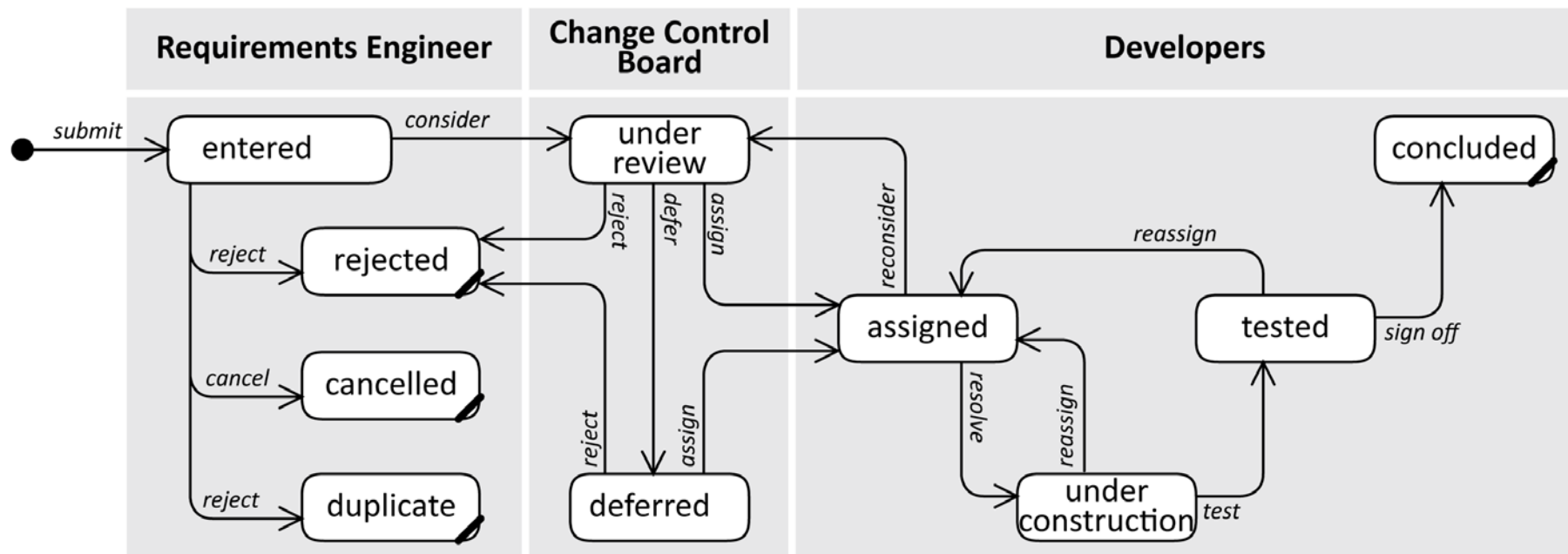
Requirements Creep

- **Also called „Scope Creep“ or „Feature Creep“.**
 - Over time, more and more features are included in the scope of the project so that it never reaches the end.
 - Empirical studies show that the yearly rate of changed and/or added requirements („churn“) may be around 3-10%.
 - Older studies suggest that by the deployment, 25% of all requirements have changed (Capers Jones, 1994).
- **Usually, this is a continuous process, where people change the requirements process in an uncontrolled way.**
 - Sometimes, however, requirements are also collected in several places. When a new collection is „discovered“, sudden changes to the requirements specification may result.
- **Generally, this is a process problem and may be overcome by a change process.**

An Industrial Change Control Process

- This activity diagram describes the process of entering and managing requirements over a prolonged period.
- It could equally well be used for handling bugs/issues.
 - In order to allow for names, final states are marked with a small diagonal slash instead of the proper notation.

AD Requirements Change Process



Analysis Paralysis

- Sometimes, requirements elicitation and/or elaboration drags on and on, without the completion ever getting closer.
 - Most of the tax software in Germany was created in the 1960s and 1970s in Cobol and Assembler running on mainframe computers.
 - The FISCUS project started 1991 with the intention of replacing all of this software over the course of a decade. After 10 years without adequate results, the project was reshaped (“fiscus GmbH”), and very nearly aborted in 2005.
 - By then, the project had delivered *“50,000 pages of documentation and 1.6 Million Lines of mostly useless code”*. Depending on the source, estimates of the cost vary between 250-900, 330-900, and 500 mio€, plus 4.5 bn€ in unclaimed taxes.
- Common reasons for such phenomena include uncertainty and fear of being made responsible personally.
- This is, ultimately, a problem of organizational culture.

Sleeping Beauty

- Sometimes, there are important requirements that are truly essential to the system, but considered uninteresting, obvious, or self-evident, and thus may be forgotten.
 - In developing the LMS case study in the RE course 2009, it went unnoticed by 25 students for two months that leases and returns will need to be recorded, somehow.
 - Obviously this is an essential requirement: without it, most of the core functions will not work.
- Possible reasons are over-motivation, or over-excitement on behalf of the development personnel that make “basic” requirements are simply forgot.



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 6.5: Individual Requirement Issues

DTU course 02264

Over-Specification

- **Some requirements may be too specific, defining a solution when all they should do is asking for one.**
 - Assume the requirement were
R1a: *“The system must provide a tape drive unit”.*
 - But is it really a tape drive unit that is required? Probably, we really need something to backup and/or archive our data.
 - So, the requirements should have just been
R1b: *“The system must provide an archiving facility for data.”*
- **While being more abstract, R1b also implies additional aspects not covered by R1a.**
 - For instance, the purpose (archiving or backup), suitable software, and so on.
 - When thinking about these, we might also become aware that we need to be specific about the amount of data we need to archive, the price, and so on.
 - Realizing this, we may improve R1 even further.
R1c: *“The system must provide an archiving facility for data capable of handling 15GB/month at a price of less than 100€/GB.”*

Over-Specification

■ Sometimes, it is hard to detect instances of over-specification.

- There may be a common and generally accepted way of achieving some goal so that everybody confuses the goal with achieving it.
- Observe that „being commonly used“ does not imply quality.

R2a: Every reader is identified to the LMS by his CPR-number.

- This is a blatant violation of data protections laws, irrespective of the fact that in Denmark everybody wants your CPR number all of the time and for everything.
- The CPR number certainly identifies the reader, but it also does identify the reader in many other ways and contexts, which is strictly unwanted.

R2b: Every reader is identified to the LMS by an unique identifier.

R3a: Every user is authenticated by login/password.

- But what about personal presence? What about young readers?

R3b: Every user is authenticated before he can do reader actions.

Premature Commitment

- **Model based specification has an inherent tendency towards specifying solutions rather than problems.**
 - Many people with a technology background find it hard to step into the shoes of people without this background and take on their perspective, temporarily.
- **There are several methods we have considered that contribute to address this premature commitment.**
 - Creating Personas helps to understand end-users.
 - Talking about goals rather than features helps understanding stakeholders.
 - Explicitly assigning layers to features, goals etc. highlights gaps and clutter.
 - Social interaction (i.e., anything from acting out to formal inspections) adds new perspectives.
- **All of these, however, are not sufficient. We need to constantly monitor our activities, and take a step back when necessary.**

- **Natural language has many advantages as a language for specifying requirements.**
- **Possibly its most important benefit is its expressiveness: there is no other language that is more expressive (for this purpose).**
- **However, this very power is also the greatest weakness of natural language: natural languages are ambiguous, and prone to errors.**

Detecting Ambiguity in Prose (1)

- There are many heuristic techniques that help finding ambiguity in (written) text.
 - They all imply considerable effort, but recall that putting in effort up front pays later on.

H1: Prose Review

- Find people with different backgrounds (including software people, domain specialists and user communities).
- Make sure, they're independent (i.e. not the authors).
- Ask, what would happen to the structure and behavior of the system, if a requirement were removed.
- Check references for clarity (e.g. *"update the field"*... which field? *"the system will then"*... when?)
- State a requirement in two different ways (possibly, one of them formal) and check whether people understand both statements as being identical.

Detecting Ambiguity in Prose (2)

H2: Effort Estimation Convergence

- Find a set of experienced, competent professionals.
- Pick a requirement or set of requirements.
- Ask the experts to estimate effort, cost, and duration of implementing the set of requirements.
- If the estimates vary greatly, ambiguity might be the cause.
 - Small to medium sized differences are likely.
 - If only a few estimates differ, it might be individual variations.
 - Estimates that differ by a factor of 10 or more and that have been estimated by several people suggest widely different interpretations and, therefore, the presence of ambiguity.

H3: Memorization

- Ask different people to memorize a specific issue (e.g., a requirement) and then later ask them to recall that issue verbatim.
- Parts that were not remembered well by the participants are likely to be places where meaning is not clear and therefore a source of problem statement ambiguity and/or vagueness.

Detecting Ambiguity in Prose (3)

H4: Shift emphasis

- Systematically shift emphasis of a requirement statement and check whether the meaning changes.

- So we will get

"Mary had a little lamb..."

it was hers, not someone else's.

"Mary had a little lamb..."

but she doesn't have it anymore.

"Mary had a a little lamb..."

just one, not several.

"Mary had a little lamb..."

it was very, very small.

"Mary had a little lamb..."

neither a goat nor a chicken.

"Mary had a little lamb..."

but John still has his.

Detecting Ambiguity in Prose (4)

H5: Use Synonyms

- Systematically replace keywords by synonyms and check whether the meaning changes.
- Here are some synonyms for “had” and “lamb” from the dictionary:
 - Had → Held in possession, acquired, accepted, marked or characterized by, held in a position of disadvantage, tricked or fooled, beget, ate, ...
 - Lamb → A young sheep, a gentle person, a pet, a person easily cheated or deceived (esp. in trading securities), ...
- Thus we may get
“Mary had a little lamb.” → “Mary conned the trader.”

H6: Shift Context

- Add another phrase and see if the meaning of the first one changed.
- So we may get
“Mary had a little lamb.” → “Mary had a little lamb and John had a lot of pasta.”

Acceptance tests reduce vagueness

- **Even if a requirements is expressed with all due diligence, it may still be unclear, just when a system satisfies this requirement. In other words, the requirement is vague.**
- **The main measure against requirements vagueness are acceptance criteria (“acceptance tests”) for each requirement.**
 - Acceptance criteria must be explicit and operational, that is, executable in a repeatable and predictable way by man or machine.
 - The most reliable way to achieve this is by writing test cases: they can be run at virtually no extra cost as often as we like.
 - However, writing them is expensive, and whenever the system changes, test cases must be changed with it.
 - The fastest and (initially) cheapest way to establish acceptance tests is manual testing.
 - However, this is inherently unreliable and soon becomes tedious (and thus error-prone), and expensive.

Informal Acceptance Tests

- **Any acceptance test consists of three parts:**

- the trigger or pre-condition, and possibly some parameters;
- an operational procedure or action to be tested
- the expected result, side effect, or post-condition.

R7: Librarians may remove or deactivate entries to the wish list.

T7.a: **Trigger:** -

Action: 1) log in as reader, open item on wish list, log off
2) log in as librarian, open item from wish list, delete, log off
3) log in as reader, look for same item from wish list, log off

Outcome: On second log in, the item shall not be found any more.
Instead, a message shall appear indicating what happened.

- **Very simple actions might testing for properties indirectly. This can be cheap and effective.**

R6: The application shall not contain absolute file paths.

T6.a: **Trigger:** -

Action: 1) Deploy application as self-contained jar to two different machines running WinXP and Linux.
2) Start the application, load an example, modify it, and save it again.

Outcome: no errors

More Variants of Acceptance Tests

- **Test cases may test several requirements collectively when appropriate.**
 - **R4a:** Every Tuesday at 8.00, LMS creates a list containing all the media that have been due in the preceding 7 days.
 - **R4b:** For all items in this list, a reminder is issued to the respective readers.
 - **T4.1:**

Trigger: Tuesday, 8.00

Action: Compare a manually created list with the system output.

Result: List of all overdue media items, grouped by lender.

- **Instead of trigger and outcome, pre- and postconditions may be used, typically expressed in terms of a system state.**
 - **R5:** The fee for late returning is computed based on the lenders' status.
 - **T5.1:**

State: L has expired 2 days ago, R has status „Proust“, M is currently reserved.¹

Action: terminate L

State: $R.fees' = R.fees + fee(„Proust“, 2)$ ²

¹ Lease L refers to medium M leased by reader R.² x' marks the next state of a variable x.

Semi-Formal Acceptance Tests

Adding formality supports the transition to coded test cases.

Case	Pre-condition or Variables	User Actions or Parameters	Post-condition or Results
T4.1	R [logged in] M [lendable]	R searches the catalog for M, R reserves M	M [reserved for R]
T4.2	R [logged in] M [lendable, not lent]	R scans M, R leases M	M [leased by R], appropriate record written
T4.3	R [logged in] M [leased by R]	PL scans M, PL returns M	L(R,M) [terminated], appropriate record written
T4.4	R [logged in], M [leased by R, not reserved], L(R, M) [not expired]	R selects M from his account, R prolongs M	L(R,M) [prolonged]

- **Define abbreviations for recurring objects or variables.**
 - M : Medium
 - R : Reader
 - L(R, M) : Lease of M by R
- **Postfixing o with [α] says that o is in state α .**
 - M [leased] : M is leased
- **Use dot notation to specify individual fields or properties of objects.**
 - M.isbn : the isbn number of M
- **Priming denotes the next state of a variable.**
 - $R.fees' = R.fees + x$
- **Refer to other acceptance test cases when possible.**
 - like T2 : like the same column of row T2
 - call T2 : invocation of case T2 (conjunction of pre/post-conditions, sequencing of actions).
- **Use comma for conjunction and sequential composition.**

Limitations of Acceptance Tests

- **Besides pure functionality (i.e. features), acceptance tests may also be used to check many quality attributes such as performance or reliability.**
- **Other properties are not so easily tested, e.g.:**
 - end-to-end performance, stress resistance/resilience;
 - maintainability, code quality;
 - input validation, usability, accessibility.
- **For these kinds of requirements, alternative test methods are needed:**
 - Specialized testing tools and manual test scripts are expensive, and often require large effort to get started.
 - Web applications and RIAs may often be tested by specialized Browser-plugins, but these are usually platform dependent, i.e. not easily portable.
 - Adherence to styleguides (e.g., concerning GUI, Coding) can be checked by reviews with appropriate checklists, or specialized tools.
 - Usability may be tested by dedicated usability tests (very expensive), or expert reviews (not quite as reliable).



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 6.6: Requirement Set Issues

DTU course 02264

Feature Interaction

- A Feature Interaction occurs, if two features that are fine individually lead to unwanted, unspecified, or nondeterministic behavior, when used together.
- This problem has first been noticed in the telecom domain, but occurs almost naturally from a certain level of system complexity on.
- Theoretically, if features are specified formally, feature interaction can be detected by formal space exploration techniques such as Model Checking.
- Practically, however, feature interaction occurs in complex systems, that is, they are usually neither formally specified, nor is the state space small enough to allow exploration of a significant part of it.

Feature Interaction CD/CLIP

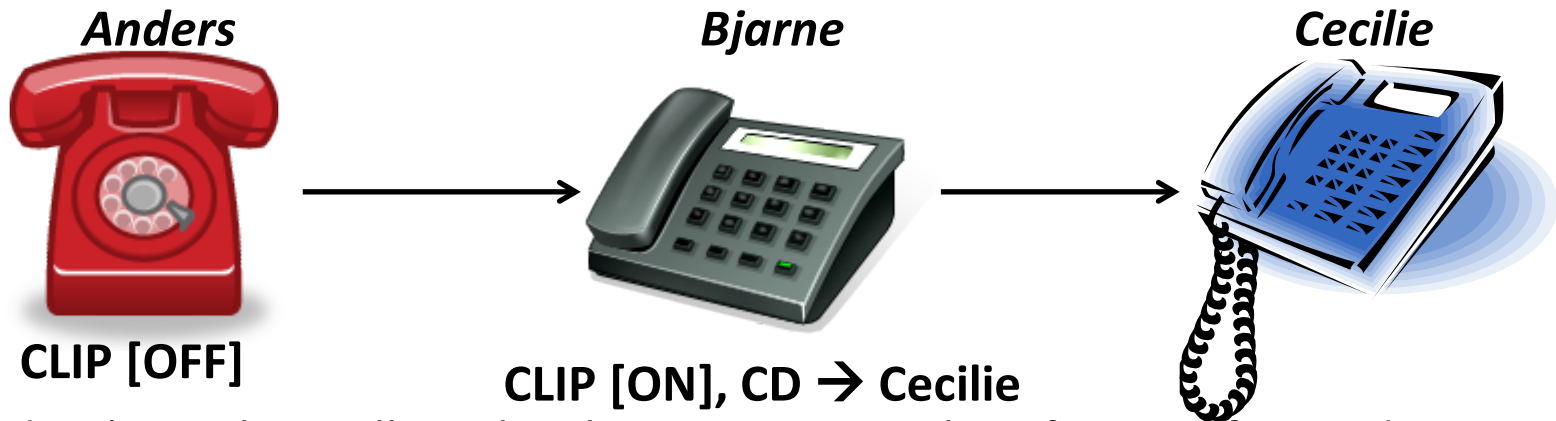
- A Feature Interaction occurs, if two features together lead to unwanted behavior, although they are fine individually.
 - This problem has first been noticed in the telecom domain, but occurs in many places.
 - Consider these two common telephone network features.

CD (Call deflection):

unanswered incoming calls are forwarded to another line which behaves just as the one originally called. Also called call forwarding.

CLIP (Calling Line Identification Presentation):

the caller's number is presented to the callee. Also called call screening.



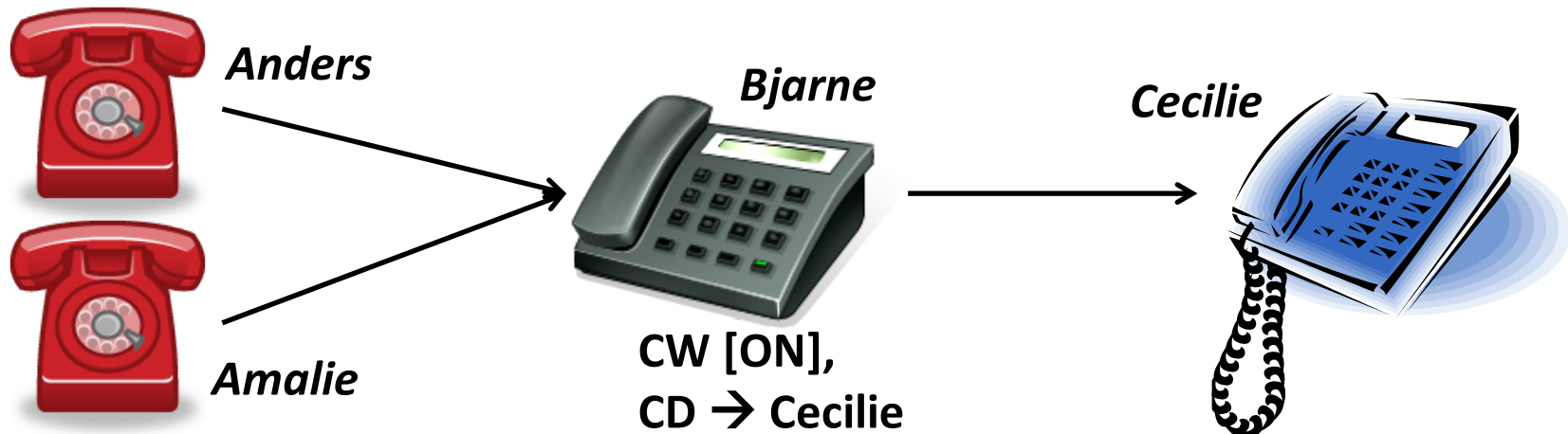
- Anders' number will not be shown to Bjarne, but if Bjarne forwards to Cecilie, will Cecilie see Anders' number or not?

Feature Interaction CD/CW

- Now consider a more complicated case, where CD and Call Waiting are activated at the same phone.

CW (Call Waiting):

If the line is busy and a new call arrives, the callee is notified and decides which call to continue.



- Anders calls Bjarne who picks up the phone. While they are talking, Amalie calls Bjarne.
- Should Bjarne be notified or Amalie's call be forwarded to Cecilie?

More FI examples in other domains

■ LMS

- Suppose the chief librarian decreases the maximum number of leased media for your account type from 15 to 10 while you have leased 12 books.
- Do you have to pay an overdraft fee? Will your account be blocked?

■ Automotive

- Suppose your car has automatic gear shift (e.g., US market, big engines) with auto-brake release (“Anfahr-Assistent”) and air conditioning.
- Suppose further you are standing in a queue on a slope. Switching on the air conditioning increases engine load (rotations), which then triggers auto-brake release, and your car rolls backwards, down the slope.

- **During requirements elicitation and elaboration, the focus is on getting the right requirements and getting them right.**
 - Relationships between the requirements are not so important, although e.g. relationships between goals and features/quality attributes may be used to justify and derive requirements.
- **When implementing and maintaining the requirements, however, the relationships are predominant as they capture knowledge needed when changing requirements.**
- **Here are some dependency types and the activities they support**
 - depends on, supports → Removal/Change
 - Rationale, conflicts/obstructs → Prioritization/Planning
 - Part of → Refinement / Elaboration

Cross-cutting Requirements

- **Requirements that affect many parts of the system, the software architecture, or just the elaboration/implementation of other features are called cross-cutting requirements.**
 - Therefore, cross-cutting requirements are difficult to add or change.
- **Most quality attributes are cross-cutting, e.g. performance or usability.**
 - But there are also cross-cutting features, e.g. Undo/Redo, or Auto-Save vs. Response time.
- **Consider a personal backup application that saves and restores files from a PC to some medium.**
 - Typical features are automatic jobs at predefined times, encryption, validation, restoring/inspection of backups.
 - Typical qualities are reliability, ease of use, and speed.
 - However, the requirement “During backup and requirement, the user can keep working with the files /drives that are being backed up” is obviously very important for many users, but it is difficult to add once the system has been designed/created.



Prof. Dr. Harald Störrle
Danmarks Tekniske Universitet (DTU)

Chapter 6.7:

Error Density and Estimation

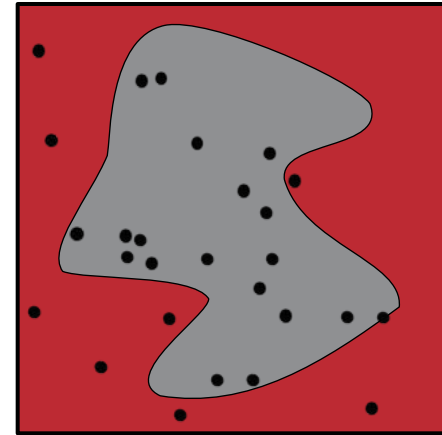
DTU course 02264

Error Density

- We can never know how many faults there are in an artefact we have not yet studied. In fact, even if we have thoroughly reviewed an artifact or tested some systems, we cannot be sure about the number of errors it contains.
- However, we can estimate the number of residual errors in a systematic way, based on stochastic sampling.
- This will help us assess the quality of a system with a given degree of certainty.
- The base argument goes like this: if k out of n items are faulty, then there will be x times as many faults in x times as many items.

Size of area by Monte-Carlo-Technique

- You can easily calculate the area of regular shapes like rectangles, but can you do it for irregular shapes, too?
- The Monte-Carlo-Technique can.
 - Place an arbitrary area of known size around the shape.
 - Drop an arbitrary number of points in this area.
 - Select an arbitrary area of known size inside the shape, and count the number of points in it.
 - Compute the ratio of points in the various areas and compare it to the ratio of the known sizes.



$$\frac{p_{\text{grey}}}{p_{\text{red}}} = \frac{a_{\text{grey}}}{a_{\text{red}}}$$

$$a_{\text{red}} \frac{p_{\text{grey}}}{p_{\text{red}}} = a_{\text{grey}}$$

where

a_{red} = area of red rectangle

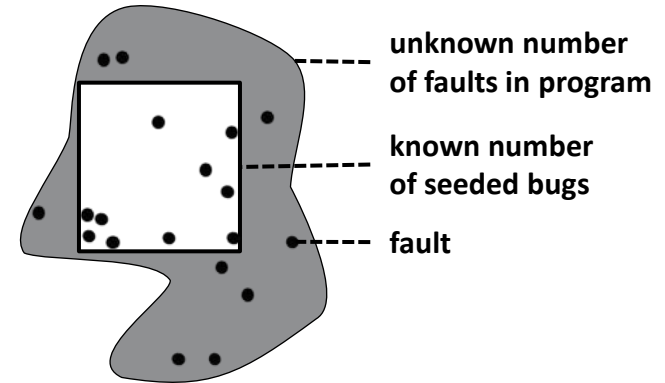
a_{grey} = area of greyshape

p_{red} = number of points in red rectangle

p_{grey} = number of points in grey area

Monte-Carlo estimation of residual faults

- We can apply this technique to estimate fault density.
 - Insert a known number of random bugs (“seeding” or “injection”).
 - Test and record the faults found.
 - Remove all found faults.



- **Example**

- Suppose there we seed 10 faults.
- During test, we find 120 faults, 6 of which were seeded.
- We estimate that there is a total of $10 \cdot 120 / 6 = 200$ faults.
- We remove $120 + 10 - 6 = 124$ faults.
- The software is shipped with 76 remaining faults.

$$\frac{\text{detected seeded faults}}{\text{detected faults}} = \frac{\text{seeded faults}}{\text{total faults in system}}$$

$$\text{total faults in system} = \text{seeded faults} \frac{\text{detected faults}}{\text{detected seeded faults}}$$

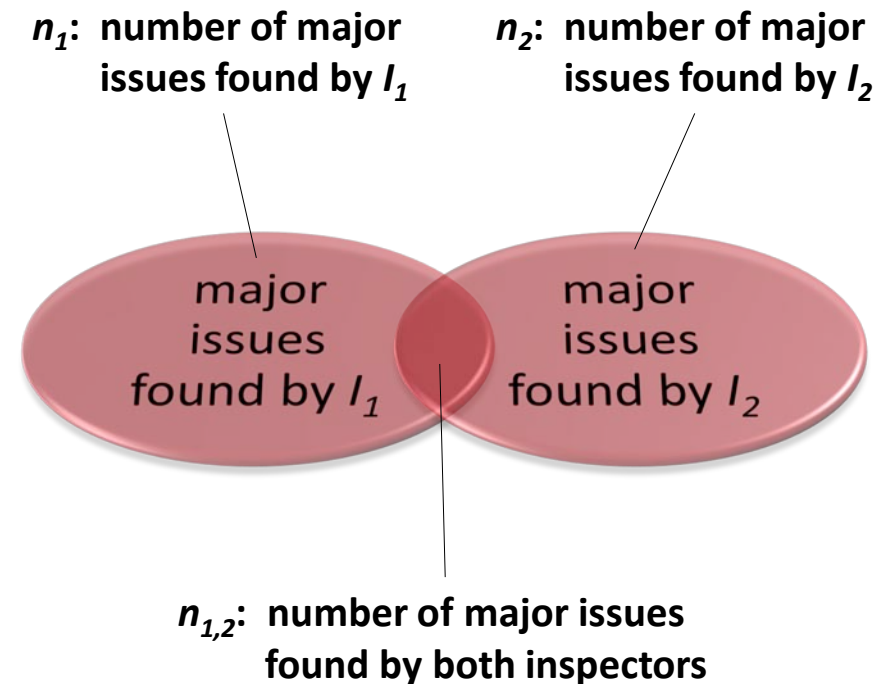
$$\begin{array}{r} \text{total faults in system} \\ - \text{detected faults} \\ - \text{seeded faults} \\ + \text{detected seeded faults} \\ \hline \text{faults delivered} \end{array}$$

Limits to the Monte-Carlo-Approach

- **Enchanting as it may seem, the Monte-Carlo-approach does have its limits.**
 - Faults have different severity: 100 minor bugs are less important than one show-stopper.
 - Faults may hide other faults so that removing faults increases their number, and sometimes fixing a mistake introduces a new one.
 - Some faults are easily found (e.g. the classical one-off-mistake), while some are difficult to track down.
 - Some bugs occur only rarely or in special situations (e.g. race conditions).
- **Still, there is no better and more proven way to estimate residual errors.**

Error Estimation by Finite Sampling

- Assume that two inspectors I_1 and I_2 find n_1 and n_2 major issues when preparing for an inspection. To some degree, their findings overlap, that is, the number of issues they both find $n_{1,2} > 0$.
- Then we can estimate the true number N of major issues based on the assumptions that
 - the two inspectors are stochastically independent, and
 - all errors are found with the same probabilities p_1 and p_2 by inspectors I_1 and I_2 .



Error Estimation by Finite Sampling

- Under these assumptions, each of the n_2 errors found by I_2 will be independently found by I_1 with probability p_1 .
- Because I_1 found $n_{1,2}$ of those n_2 errors, it is reasonable to estimate \hat{p}_1 for p_1 with

$$\hat{p}_1 = \frac{n_{1,2}}{n_2}$$

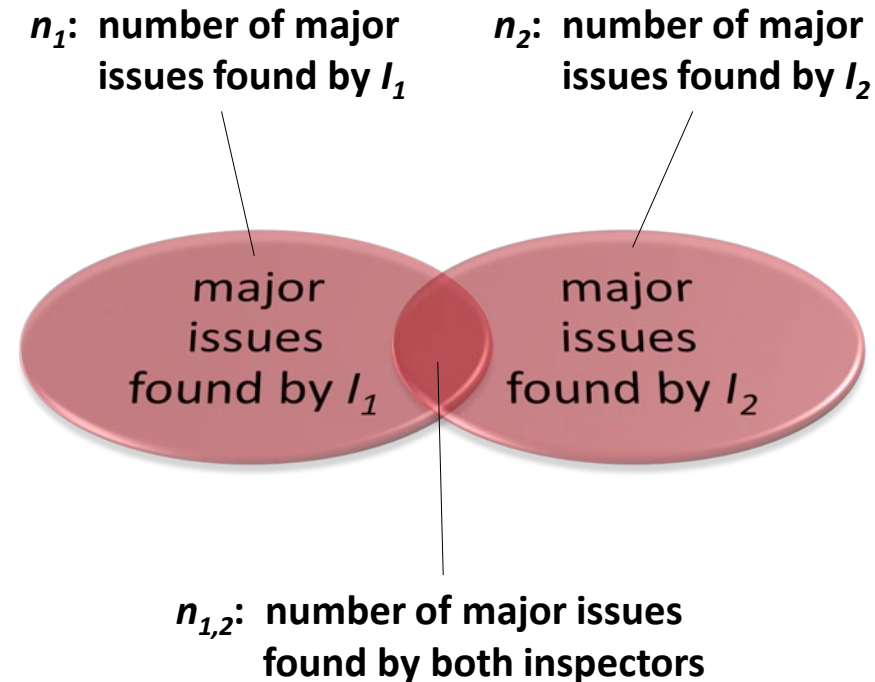
On the other hand, since I_1 found n_1 of the N errors, it is also justified to assume

$$\hat{p}_1 = \frac{n_1}{N}$$

and thus

$$\frac{n_{1,2}}{n_2} = \frac{n_1}{N} \Leftrightarrow$$

$$N = \frac{n_1 * n_2}{n_{1,2}}$$



Problems & Solutions

- **The two inspectors find no overlap.**
 - We assume there is one major issue they both found and estimate $N=n_1*n_2$.
- **There are three inspectors but only two of them overlap (I_1 and I_2 , say).**
 - As a simple heuristic, assume I_2 and I_3 (two non-overlapping inspections) to be just one inspection and estimate

$$N = \frac{n_1 * (n_2 + n_3)}{n_{1,(2+3)}} .$$

- **There are three inspectors and they all overlap.**
 - The proper solution is complicated. As a simple heuristic, apply the method to all pairs and take the maximum of the estimates.

Applications of error injection

- **The error injection technique can be used in multiple ways in the area of requirements engineering quality assurance.**
 - Inject errors to any of the artifacts created and keep track of where they were injected. This can be applied for various error levels.
 - For instance, for personas, we could
 - add a mistake to an existing persona or persona set;
 - remove an important aspect from a persona;
 - remove a necessary persona or add a inappropriate/superfluous/erroneous persona.
- **Comparing the number of seeded errors and found errors as shown above, we can estimate**
 - the number of residual errors in the artifact, and thus
 - the thoroughness of the inspection.
- **We could also trace effective and ineffective guidelines, and by reducing their number possibly reduce the overall effort.**