

Software Engineering Methodology

Lecture 7, 24.2.2014

Kari Systä

Today

- Summary of Lean and Kanban
- Continuous Deployment, DevOps (we did not have time to cover it last week)
- Quality assurance
 - Inspections and reviews
 - Testing

This weeks weekly exercise

- Tue 11.02.2014 at 10-12 and 12-14
- Wed 12.02.2014 at 12-14
- Thu 13.02.2014 at 12-14 and 14-16

- In TC217!

Initial content of lectures

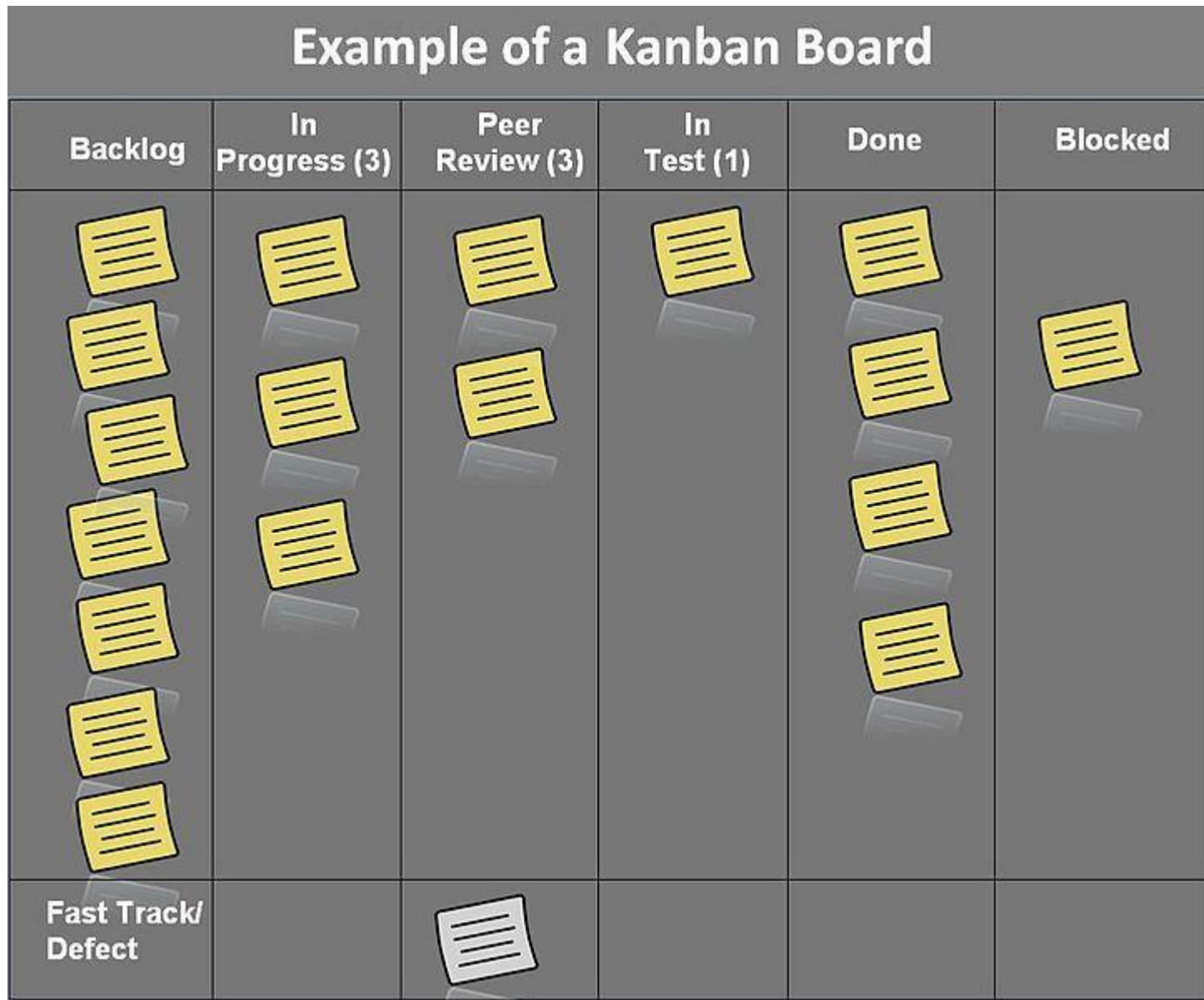
- Introduction
- Life-cycle models, their background
- Project management, product management, project planning – in general management aspects
- Scrum in details
- Requirement elicitation, requirement management, requirements prioritization
- New trends: Lean, Kanban, Lean Start-up,
- Continuous deployment, DevOps; **Review practices, testing and quality assurance (TIE-21200 will go deeper)**
- Another perspective to quality: “Quality systems” and process improvement
- Version management, configuration management, continuous integration
- Architecture issues, role of architect, architectural quality attributes, product families, (TIE-21300 will go deeper)
- Embedded and real-time systems (other courses will go deeper)
- Safety-critical and dependable systems
- Effort estimation
- Software business, software start-ups
- Recap

Summary of lean

- Avoid waste
- Make progress and state visible
- Deliver fast and continuously
- Build quality in & fix problems immediately (stop the work)
- "Japanese dictionary"
 - continuous improvement (kaizen)
 - relentless reflection (hansei)
 - thoroughly understand the situation (genchi genbutsu)
 - Decide slowly; implement rapidly (nemawashi)
 - Level out the workload (heijunka)

Summary of Kanban

- Visualise
- Limit Work-in-progress (WIP)
- Manage flow
- Make policies explicit
- Implement feedback loops
- Improve collaboratively, evolve experimentally (using models and the scientific method)

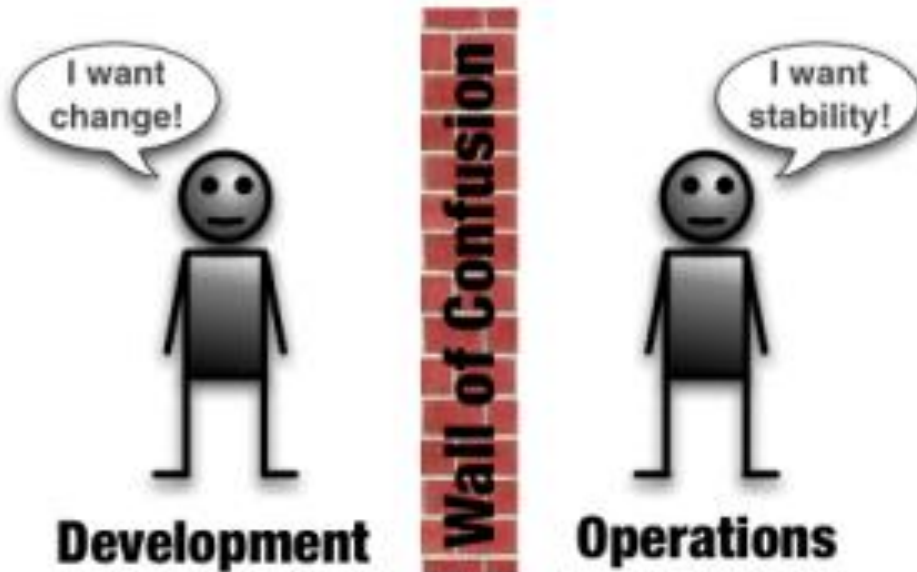


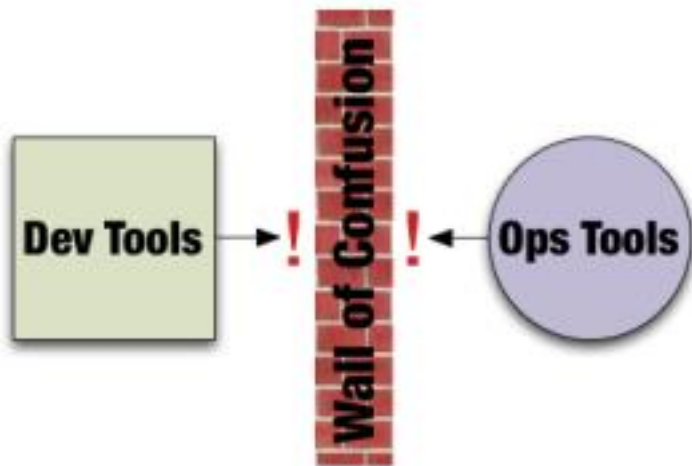
- Have every problem once
- Stop the line when anything fails
- Fast response over prevention

DevOps

(<http://dev2ops.org/2010/02/what-is-devops/>)

- DevOps is a response to the growing awareness that there is a disconnect between what is traditionally considered development activity and what is traditionally considered operations activity. This disconnect often manifests itself as conflict and inefficiency.
- Wall of confusion

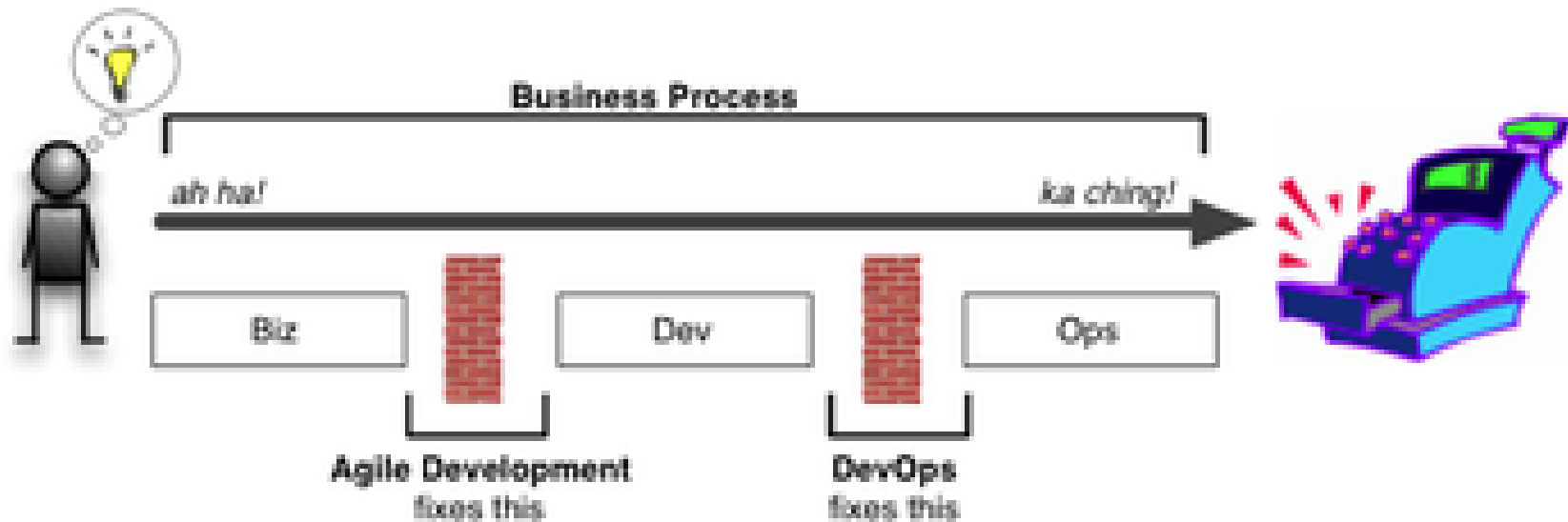


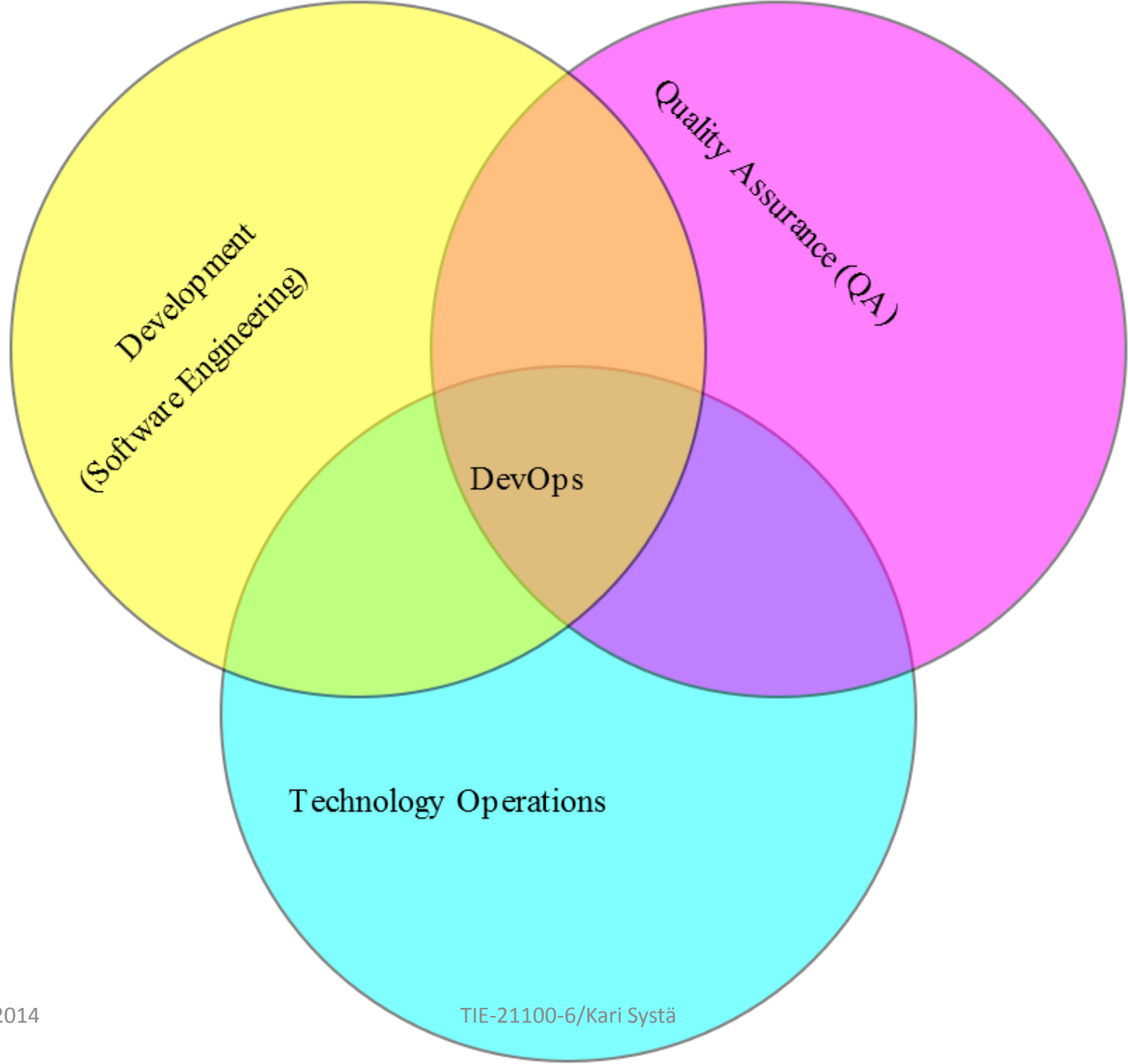


=



The lifecycle





Content of DevOps

- Typically uses agile development processes
- Increased rate of production releases
(Continuous development)
- Common tools
- Use of virtualized and cloud infrastructure
from internal and external providers
- Increased usage of data center automation
and configuration management tools

Material

- Liker: The *14 Principles* of the *Toyota Way*: An Executive Summary of the Culture Behind TPS
<http://icos.groups.si.umich.edu//Liker04.pdf>
- Alan Shalloway, Demystifying Kanban
<http://www.netobjectives.com/files/resources/articles/Demystifying-Kanban.pdf>
- Aspects of Kanban
<http://www.methodsandtools.com/archive/archive.php?id=104>
- <http://www.netobjectives.com/blogs/real-differences-between-kanban-and-scrum>
- Presentation by Eric Ries:
<http://www.gov2summit.com/gov2009/public/schedule/detail/10560>
- DevOps: <http://dev2ops.org/2010/02/what-is-devops/>
-

REVIEWS AND INSPECTIONS

Terminology

Inspection = tarkastus

- Internal event in the project - not strictly tied to project schedules (next phase may start)
- Sole purpose is in detecting defects for early correction
- Relatively small amount of work under inspection
- The whole material (to be inspected) is scanned through
- More "formal", diary/log/minutes is kept
- Documents, code, prototypes,...

(Technical) Review = katselmointi, katselmus, tekninen katselmus

- Formal event to check that a milestone have been reached; makes a milestone in a project visible (go / no-go)
- The entire phase product
- Number of participants can be large and different stakeholders less formal, diary/log/minutes maybe not kept, just "scanning" material.

Walkthrough (, walk-thru) = läpikäynti

- Informal - "what the designer thinks his/her code does".

Assessment

- Usually for processes (will be discussed in next lecture)

Definitions

- IEEE 610.12-1990:

inspection = a static analysis technique that relies on visual examination of development products to detect errors, violations of development standards, and other problems. Types include code inspection; design inspection.

review = a process or meeting during which a work product, or set of work products, is presented to project personnel, managers, users, customers, or other interested parties for comment or approval. Types include code review, design review, formal qualification review, requirements review, test readiness review.

walk-through = a static analysis technique in which a designer or programmer leads members of the development team and other interested parties through a segment of documentation or code, and the participants ask questions and make comments about possible errors, violation of development standards, and other problems.

Difference between inspection and reviews

- There are two intentions
 - Inspect to find errors
 - Review to ensure that milestone has been reached, i.e., all conditions of the milestone are met
- Procedure may very similar
- The meeting in inspections is often called review

Inspections

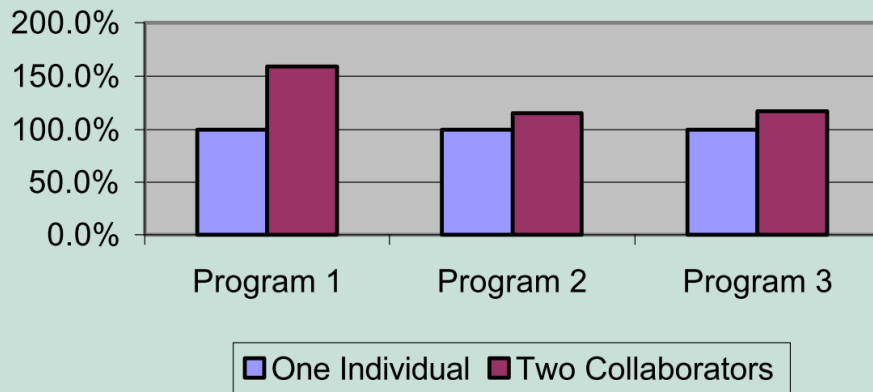
- Very efficient way to find errors
- Fagan: "Design and code inspections to reduce errors in program development", IBM Systems Journal, 1976.
- Requires effort – is an investment

Side note: pair programming

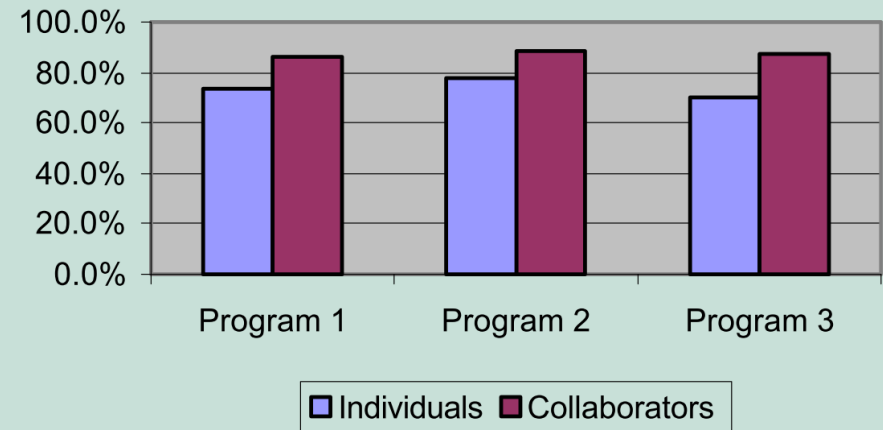
This not inspection

- Cockburn, Williams: The Costs and Benefits of Pair Programming
- Two programmers work collaboratively on the same algorithm, design or programming task, sitting side by side at one computer.
- One writes code and the other “inspects”

Relative Time: One Individual vs Two Collaborators



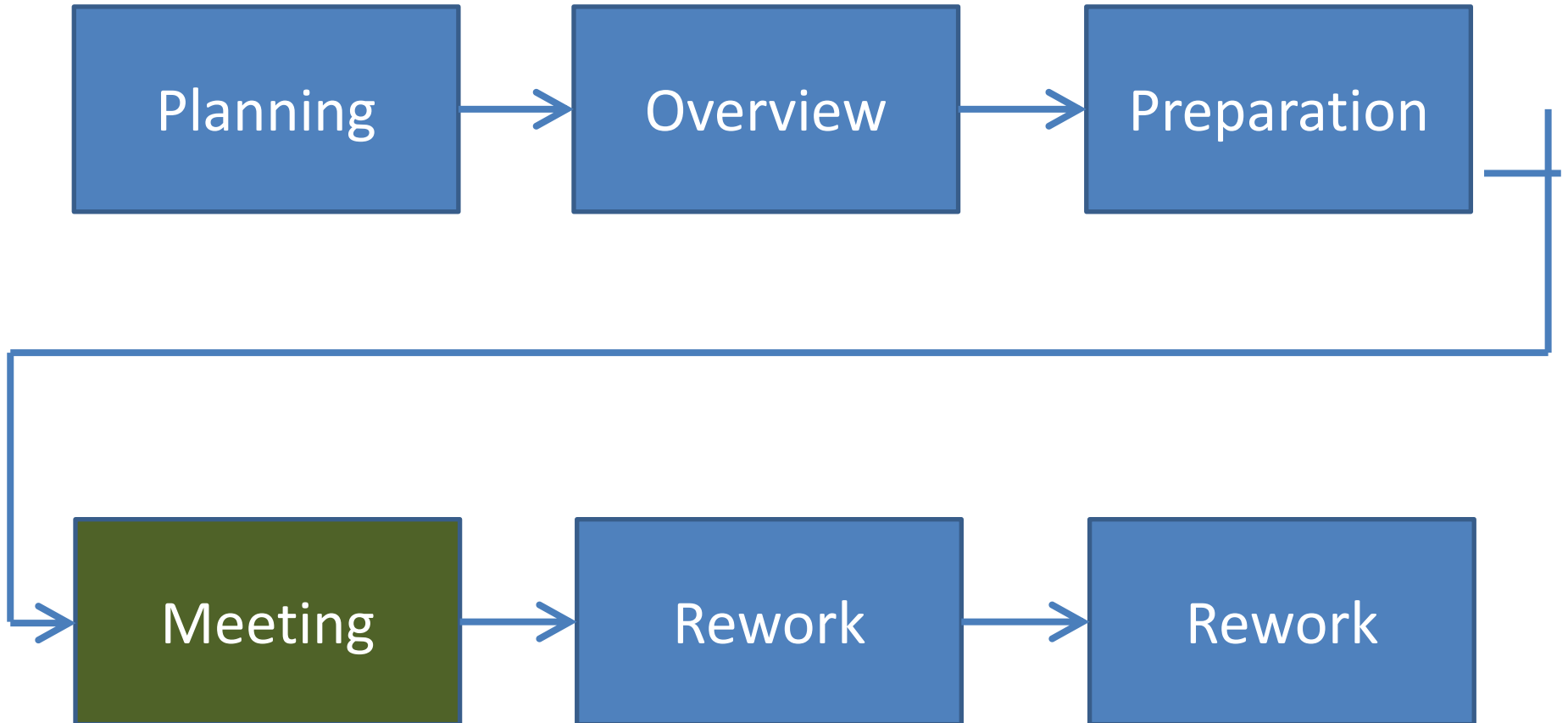
Post Development Test Cases Passed



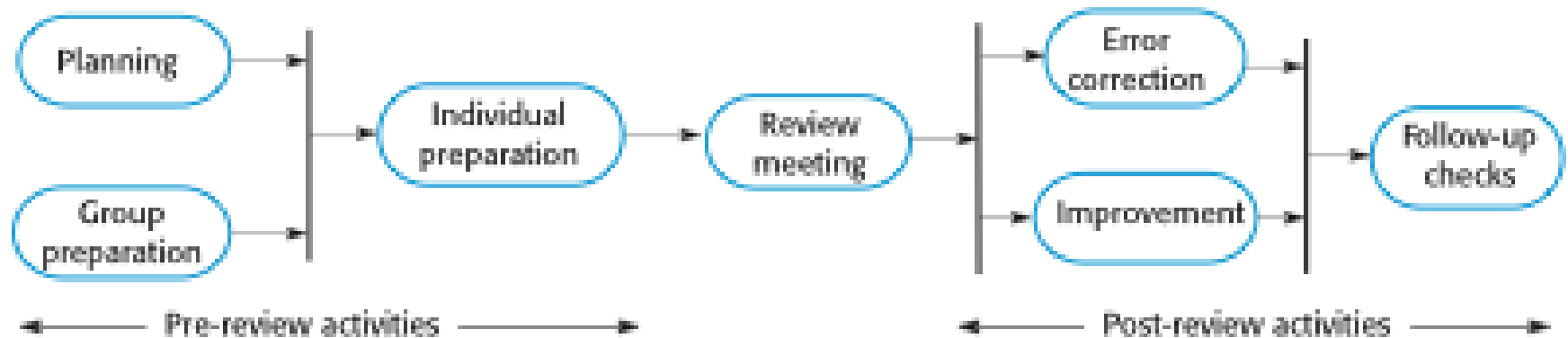
About economy of inspections

- To find defects as early as possible
- To make design, central code, documents etc as stable (and as early) as possible with the help of technical colleagues' input and experience.
- 5% to 15% of project cost (working time)
- Finds up to 80% of the defects in product (normally less, never 100 %)
- Cost effective in improving quality
- Testing is too late; all faults in specs, design documents etc. are already implemented. That is why inspections are needed and are cost effective

Inspection in practice



Sommerville drawing of the process



Roles

(<http://www.softwareengineering-9.com/Web/QualityMan/roles.html>)

Grady and Van Slack (Grady and Van Slack, 1994) suggest six roles:

- **Author or owner.** The programmer or designer responsible for producing the program or document. Responsible for fixing defects discovered during the inspection process.
- **Inspector.** Finds errors, omissions and inconsistencies in programs and documents. May also identify broader issues with the code being inspected such as lack of portability.
- **Reader.** Presents the code or document at an inspection meeting.
- **Chairman or moderator.** Manages the process and facilitates the inspection. Reports process results to the chief moderator.
- **Scribe.** Records the results of the inspection meeting.
- **Chief moderator.** Responsible for inspection process improvements, checklist updating, standards development, etc. Not necessarily involved in all inspections.

Roles (as instructed in our project course)

Moderator, coordinator, chairman, leader

- Leads the meeting; keep the discussion in defects, may even cancel the meeting at start if it seems to be worthless (e.g. poor material or poor preparation) !

Inspectors, checkers, participants

- Voice comments on the section read, and look for errors

Secretary

- Fill in defect list

Author

- Get a clear understanding about what needs to be corrected.

Good practices (from project course)

no need to ask "turn to speak"

if you don't have any comments for that section/paragraph/line, no need to say that

speak shortly (no coffee table conversations)

if you are in doubt (if there is an unclear matter), ask !

"is that an error or not ?", unclear matters should be recorded to diary/log/minutes

"a professional attitude" is the best; do it right, do it once, **don't waste time.**

Do not discuss solution

It helps a lot if the diary/log/minutes is visible to all participants all the time

- everybody can see and check what was written down
- to see if secretary did understand finding/question right
- is the speed too fast (or slow) considered to writing diary
- no need for repetition/review of diary at the end of inspection.

If target is "ready" and people are prepared well, inspection goes fast and easy.

Typical problems

- unprepared attendees ("I'm in a hurry" is no excuse !!!)
- unstable (draft) specs, interfaces in code
- inspection changes into a design meeting
- irrelevant comments
- too much material
- follow-up not performed
- "crying" author or moderator
- document prepared in isolation, hopeless quality, somebody should prechecked
- process differences
- political decisions (e.g. QA vs. delivery deadlines)
- inspections simply not performed
- disturbances (phones, people coming late or leaving early,...).

<http://www.cs.tut.fi/kurssit/OHJ-3500/2012-13/tilastoja.html>

Simple data collected from course document inspections

- duration of inspection; e.g. 1,75 h
- total inspection time (prep+insp); e.g. 22,5 h
- number of findings; e.g. 23
- number of pages in document; e.g. 37.

From these the following statistics can be calculated easily:

- error density; errors / page (e.g. 0,62)
- speed of inspection; pages / hour (e.g. 21,14)
- speed, number of findings per hour; errors / hour (e.g. 13,14)
- time for finding one error; total inspection time / number of errors (e.g. 0,98 h)
- time spent for one page; inspection time / page (e.g. 2,84 min).

Reviews and agile methods

- The review process in agile software development is usually informal.
 - In Scrum, for example, there is a review meeting after each iteration of the software has been completed (a sprint review), where quality issues and problems may be discussed.
- In extreme programming, pair programming ensures that code is constantly being examined and reviewed by another team member.
- XP relies on individuals taking the initiative to improve and refactor code. Agile approaches are not usually standards-driven, so issues of standards compliance are not usually considered.

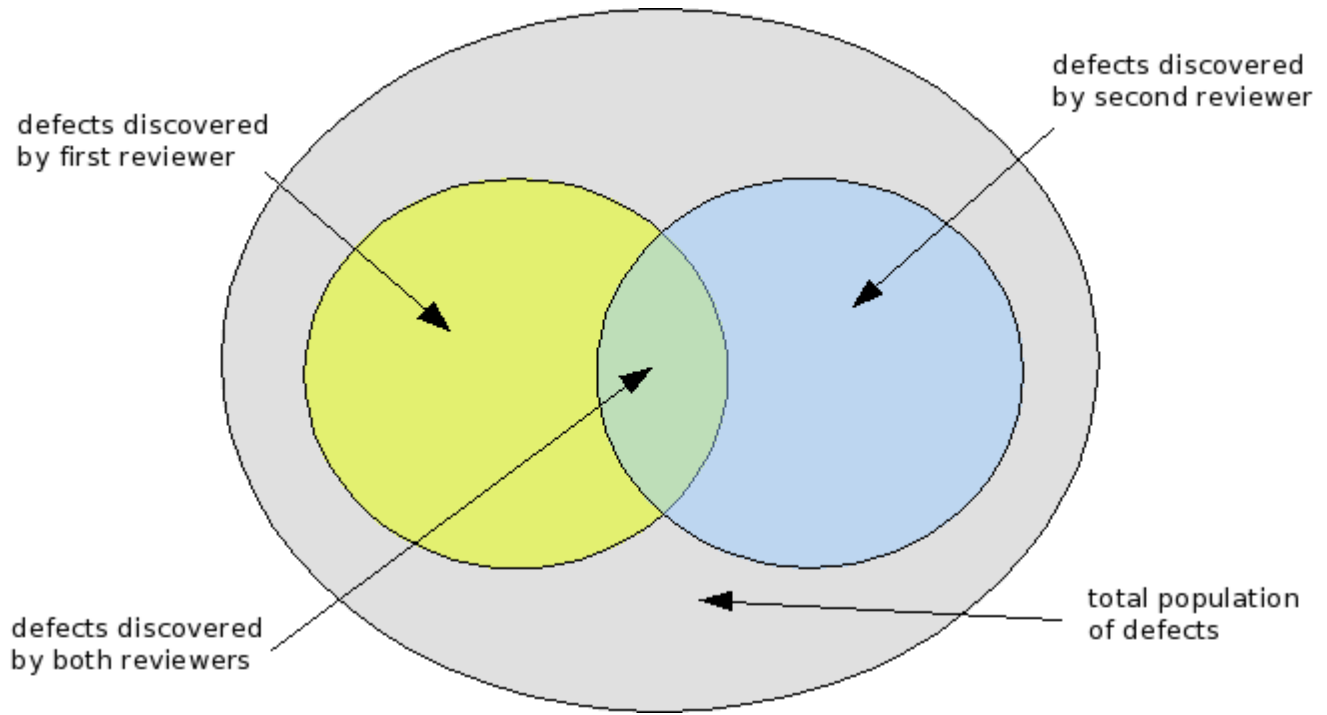
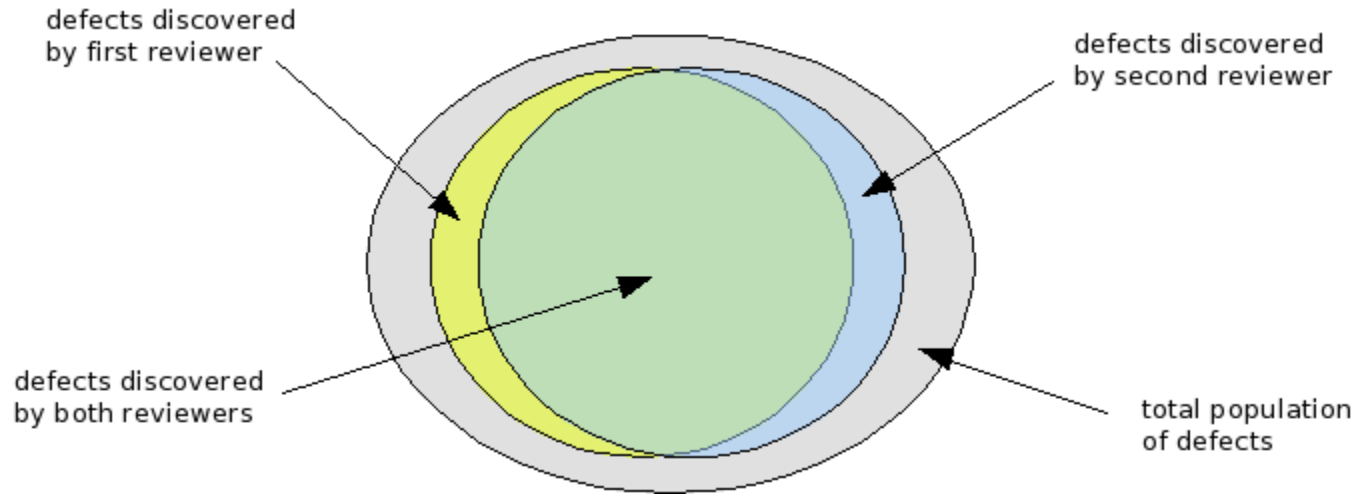
One Agile method, Feature-Driven Development, recommends use of inspections

Develop features using the following practices:

- Domain Object Modeling
- Developing by Feature
- Component/Class Ownership
- Feature Teams
- **Inspections**
- Configuration Management
- Regular Builds
- Visibility of progress and results

Lean community has developed capture-recapture code inspection

(<http://leansoftwareengineering.com/2007/06/05/the-capture-recapture-code-inspection/>)



Theory

$$N = \frac{n_1 * n_2}{m}$$

$$N = \frac{(n_1+1)(n_2+1)}{(m+1)} - 1$$

N = Estimate of total defect population size

n1 = Total number of defects discovered by first reviewer

n2 = Total number of defects discovered by second reviewer

m = Number of common defects discovered by both reviewers

n1	n2	m	N1	N2
1	10	1	10	10
2	9	2	9	9
3	8	2	12	11
4	7	2	14	12
5	6	2	15	13
6	5	2	15	13
7	4	1	28	19
8	3	1	24	17
9	2	1	18	14
10	1	1	10	10

Capture-recapture code inspection

The basic version requires four roles:

- the reviewee
- reviewer A
- reviewer B
- review moderator
- Approximately 200 lines of code will result in the highest *inspection yield*
- Code sent to sent to the two reviewers
- The reviewers bring their marked-up documents to the review meeting. Each inspector will enumerate his findings, and the group will validate each defect.

Walk-through

- More light-weight than inspections
- Informal - “what the designer thinks his/her code does”.
- The second motivation is spreading information
- Very useful for code-segment that all other developers use or are dependent on

TESTING

Testing

Disjkstra:

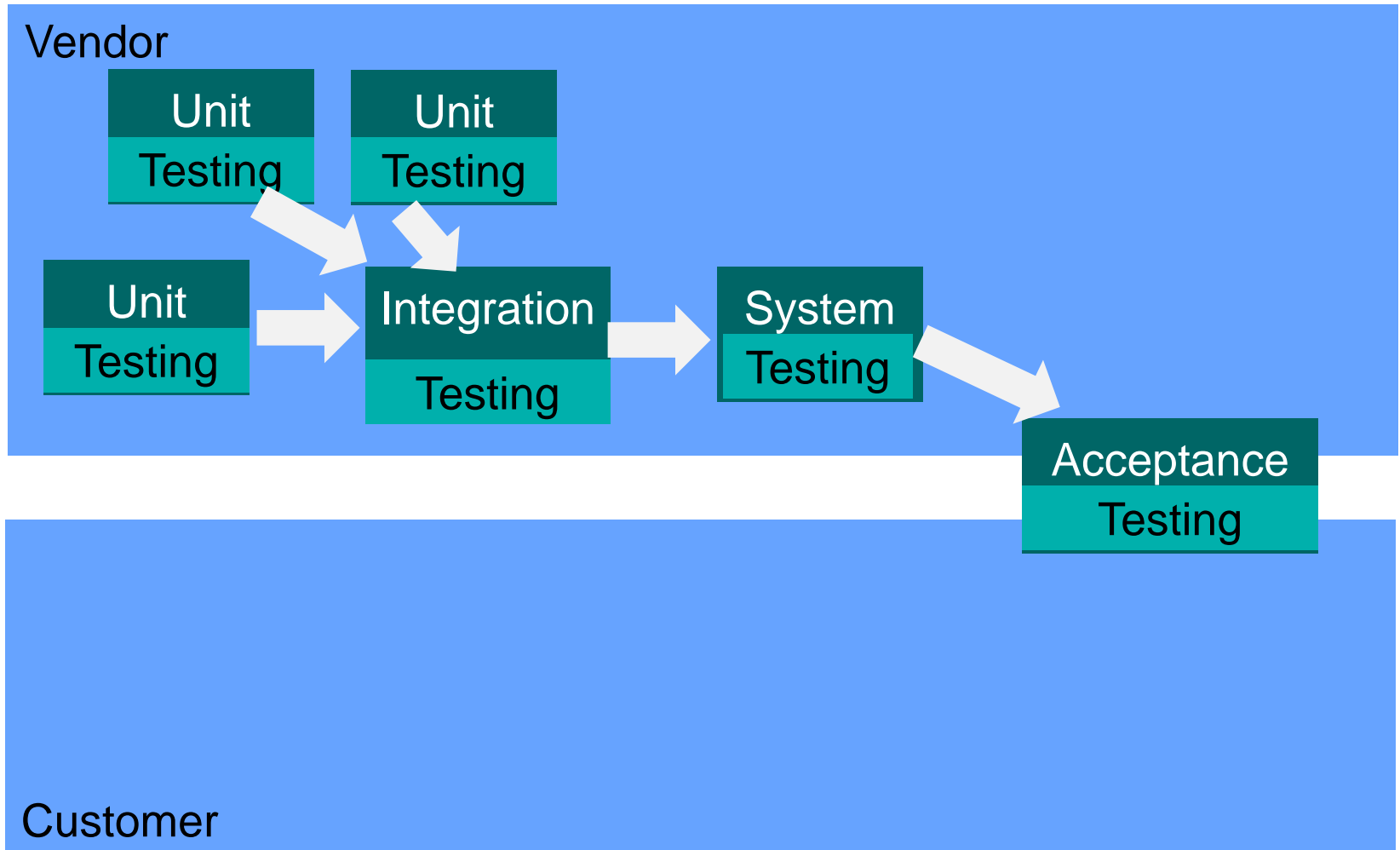
- "Testing can only show presence of errors, not their absence"

Inspections and testing both have their roles:

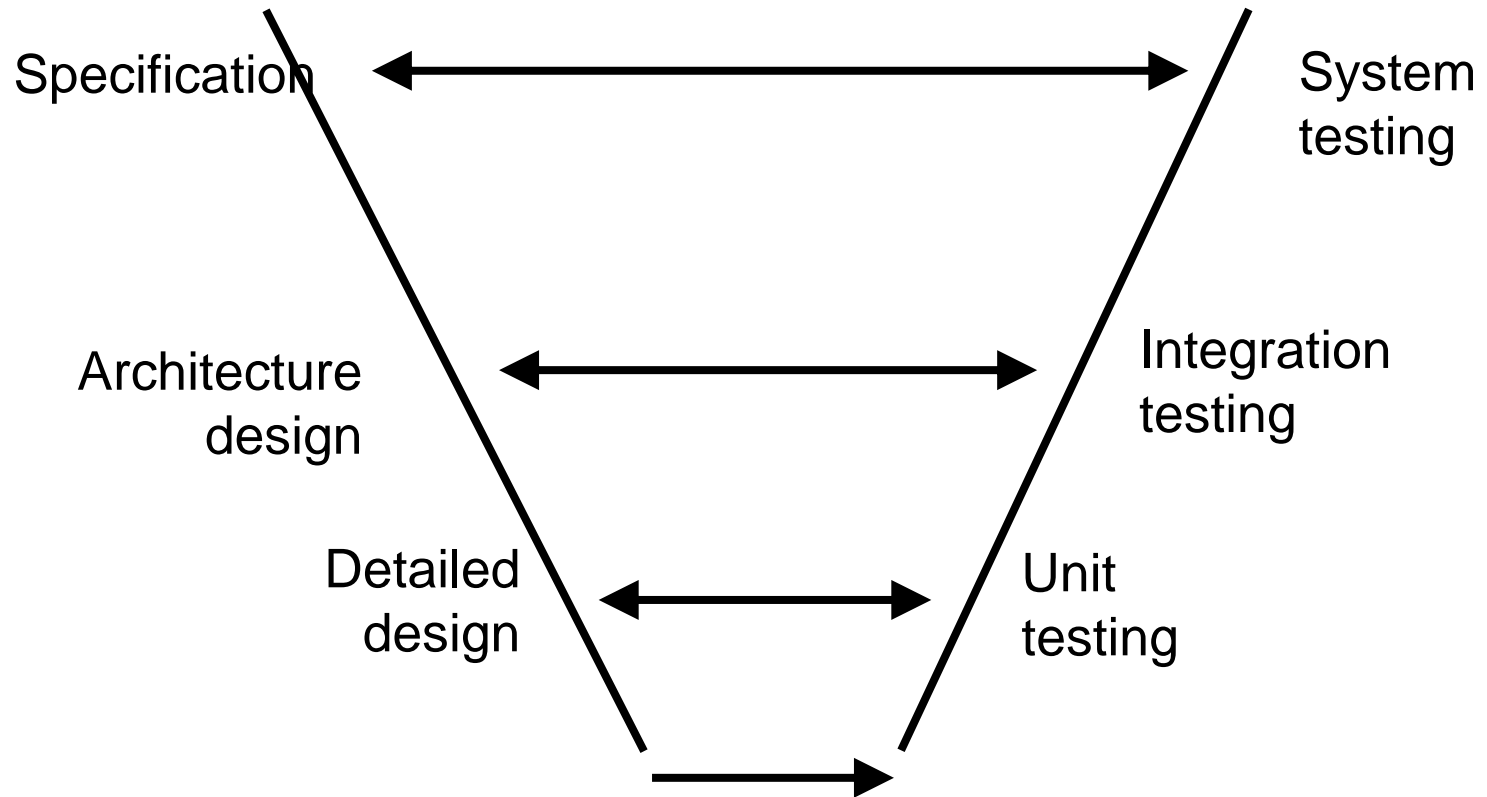
- In testing error can mask other errors
- Incomplete versions can inspected
- Inspections can consider broader set of quality attributes
- Tests are easy to repeat
- Testing can discover issues that relate timing, interactions between different parts of software, ...

- **TIE-21200 Ohjelmistojen testaus / TIE-21200 Software Testing**
- **Learning outcomes of the course (SG*)**
- Opiskelija tuntee testaamisen peruskäsitteet ja -tekniikat yksikkö-, integrointi-, järjestelmä- ja hyväksyntätestaustasolla sekä osaa soveltaa niitä ohjelmistotyössä kaikissa elinkaaren vaiheissa. Opiskelija tunnistaa sellaiset testaukseen liittyvät tehtävät, jotka voidaan joko osittain tai kokonaan automatisoida työkalujen avulla. Lisäksi opiskelija osaa käyttää vähintään yhtä automatisointityökalua.
- Students knows fundamental concepts of testing, and techniques for unit, integration, system and acceptance testing and apply these in all phases of software development. Student recognizes testing tasks that can partly or completely automated. In addition, students will learn to use at least one test-automation tool.

Testing – traditional categorization



V-moded



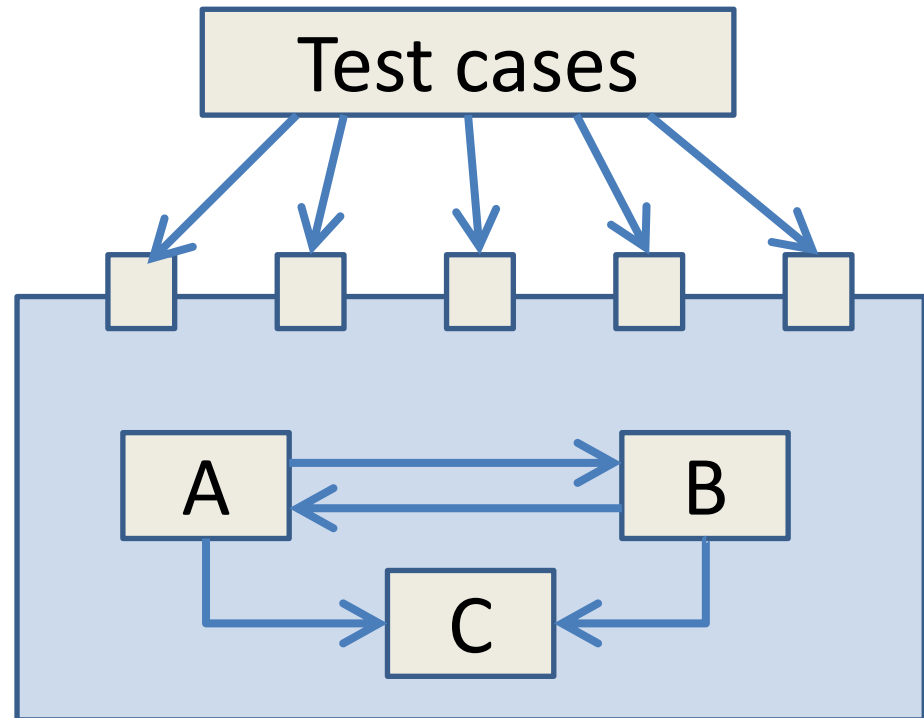
Unit testing

- Software unit is tested before it is integrated to other parts of the system
- Typically the size of tested unit is 100-1000 lines of code
- To run a unit in isolation developers need use test-beds and test drivers
- For example if the unit is a class or object, the test should
 - Call each method with a representative parameter values
 - Set and check values of attributes
 - Put object to different states

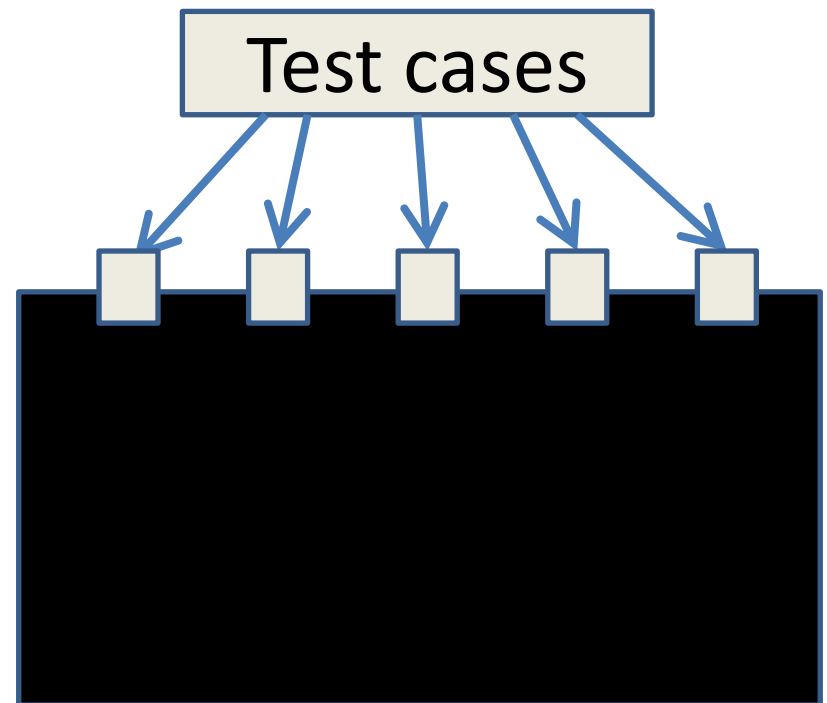
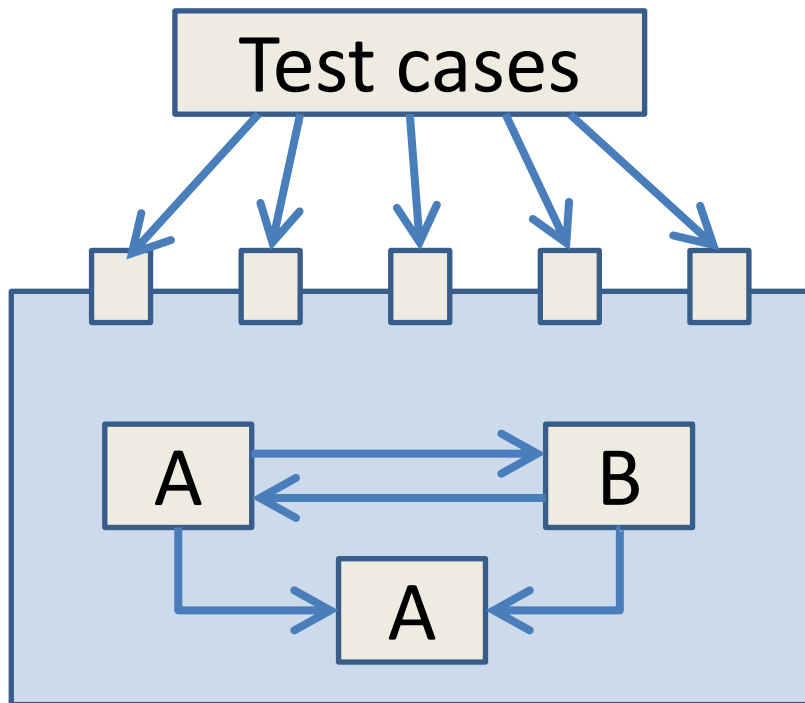
Integration testing

(Sommerville uses term Component testing)

- Test how units work together
- Usually concentrates in major interfaces in the software



Black box and white box testing



About test coverage

```
int foo (int x, int y) {  
    int z = 0;  
    if ((x>0) && (y>0)) {  
        z = x;  
    } else if (x*y > 0) {  
        z = x + 1;  
    }  
    return z;  
}
```

Code coverage

Condition/decision coverage

Parameter value coverage

...

More on these in weekly exercises

This weeks weekly exercise

- Tue 11.02.2014 at 10-12 and 12-14
- Wed 12.02.2014 at 12-14
- Thu 13.02.2014 at 12-14 and 14-16

- In TC217!

System testing

- System is addresses the complete system
- Compared to functional specification and use manual
- Acceptance testing is a form of system testing that is done together with the customer
 - Passing a. test is prerequisite for approval of the delivery

System and acceptance testing

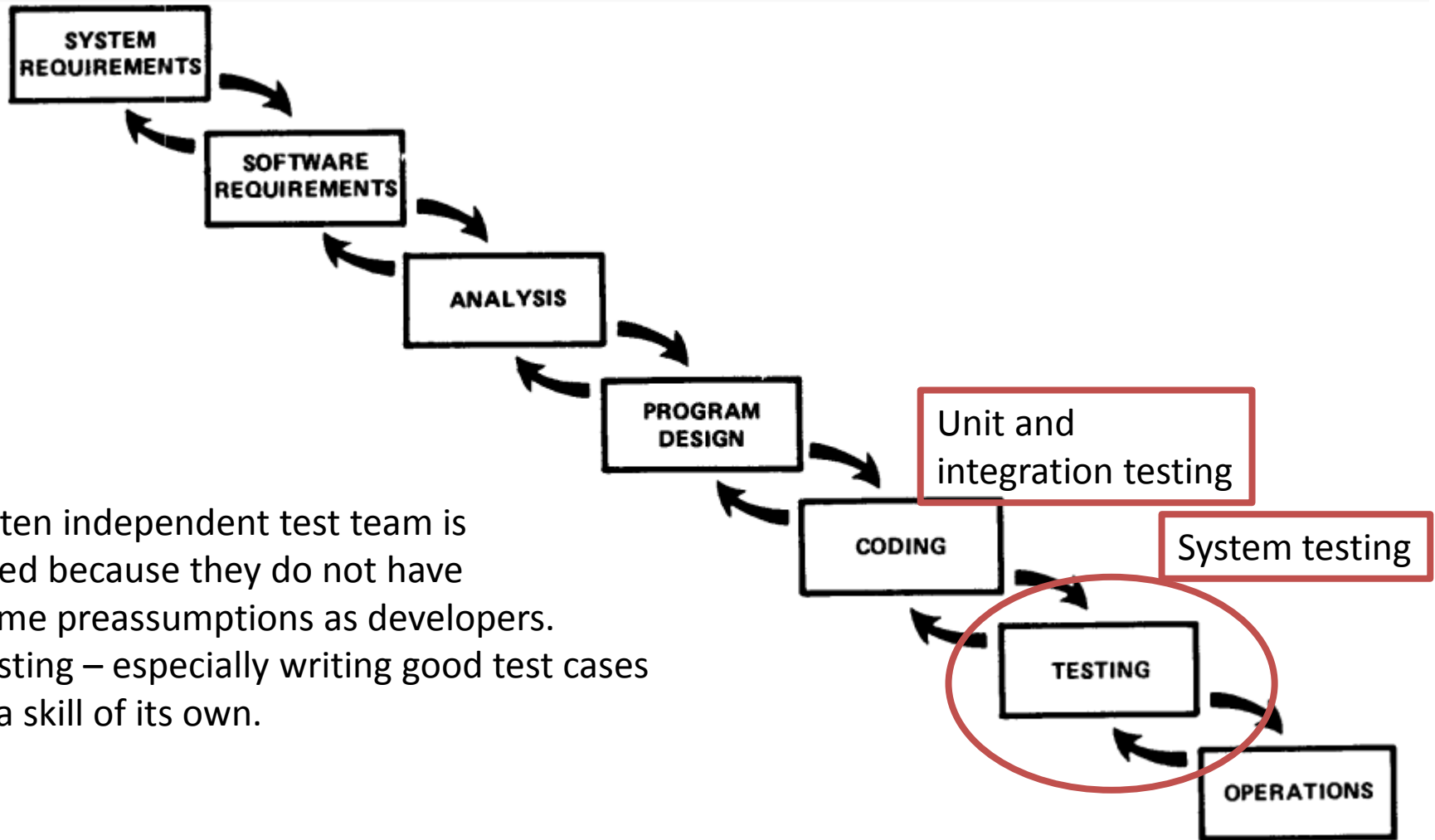
- May include
 - Field testing
 - Performance testing
 - Load testing
 - Reliability testing
 - Installation testing
 - User / Usability testing

Test planning

<http://www.softwareengineering-9.com/Web/Testing/Planning.html>

- **The testing process** A description of the major phases of the system testing process. This may be broken down into the testing of individual sub-systems, the testing of external system interfaces, etc.
- **Requirements traceability** Users are most interested in the system meeting its requirements and testing should be planned so that all requirements are individually tested.
- **Tested items** The products of the software process that are to be tested should be specified.
- **Testing schedule** An overall testing schedule and resource allocation. This schedule should be linked to the more general project development schedule.
- **Test recording procedures** It is not enough simply to run tests; the results of the tests must be systematically recorded. It must be possible to audit the testing process to check that it has been carried out correctly.
- **Hardware and software requirements** This section should set out the software tools required and estimated hardware utilisation.
- **Constraints** Constraints affecting the testing process such as staff shortages should be anticipated in this section.
- **System tests** This section, which may be completely separate from the test plan, defines the test cases that should be applied to the system. These tests are derived from the system requirements specification.

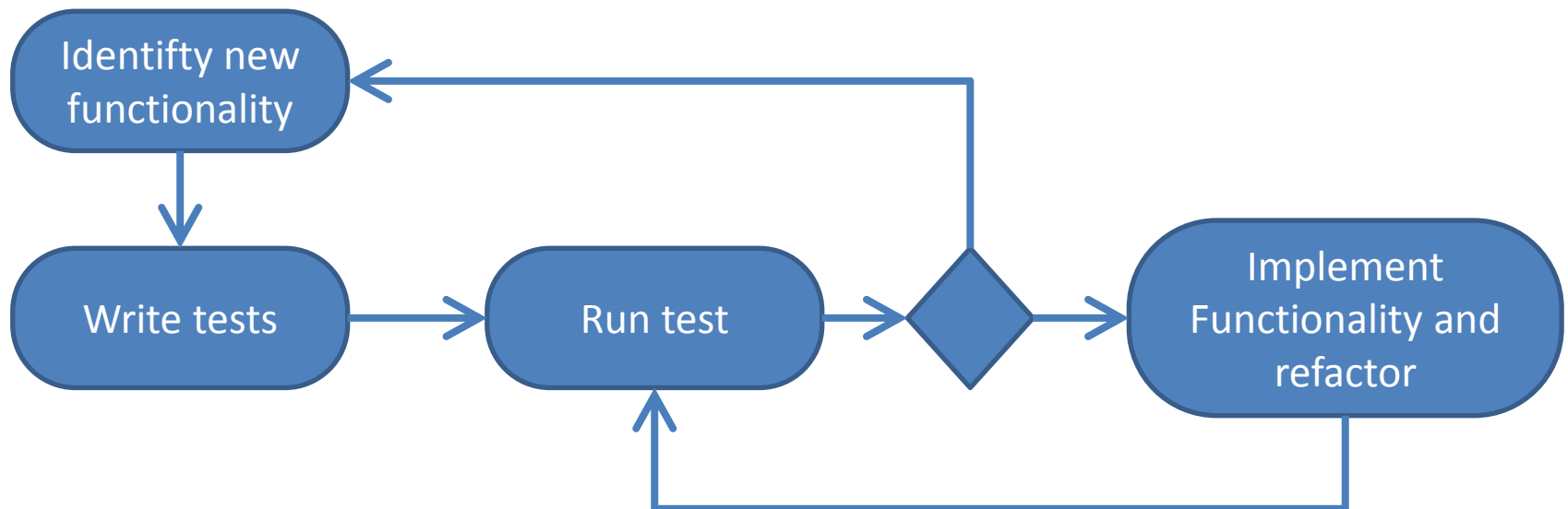
Testing and waterfall



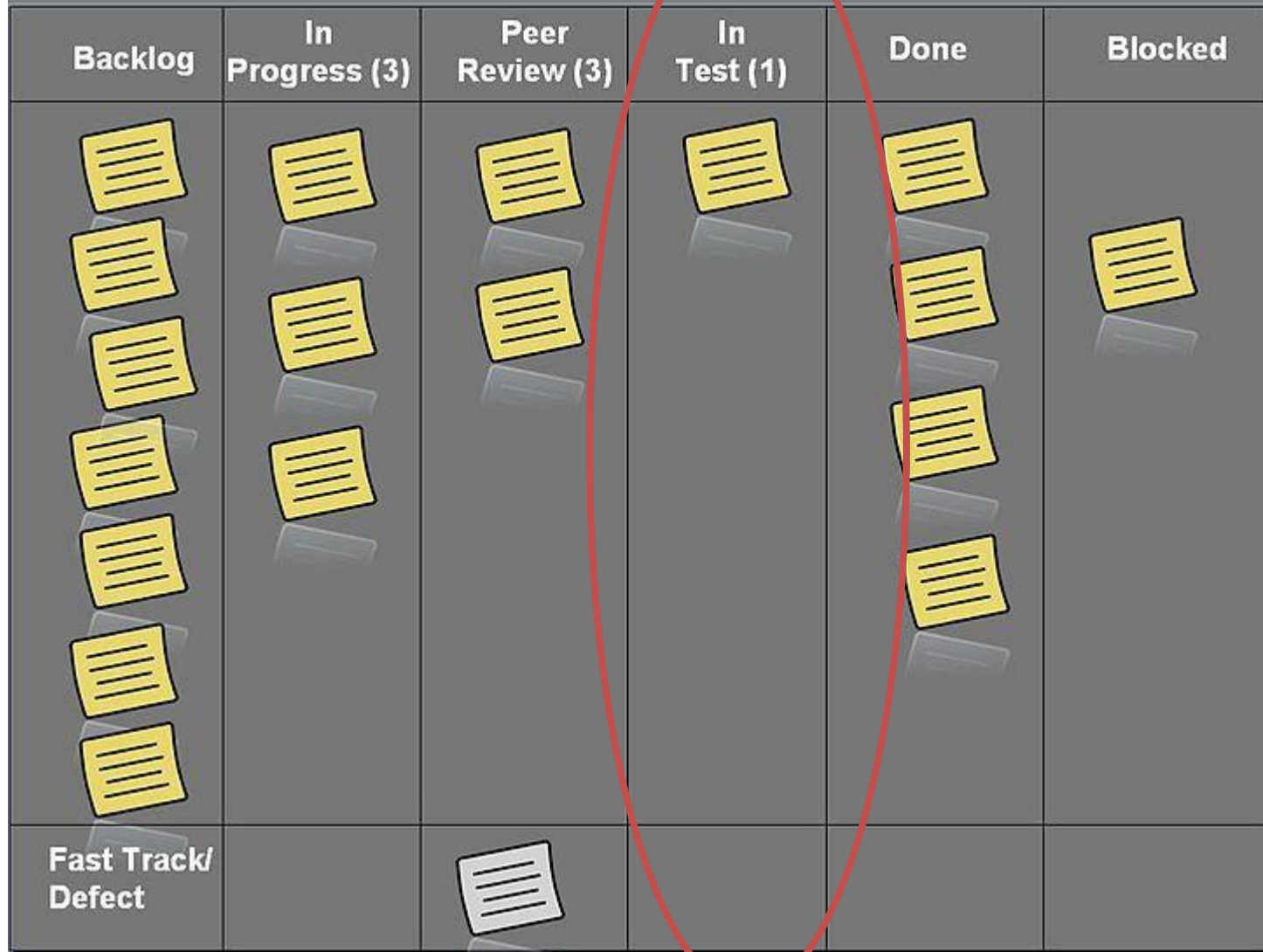
Often independent test team is used because they do not have same preassumptions as developers. Testing – especially writing good test cases is a skill of its own.

Testing and Agile

- Testing is a crucial part of agile processes
- The iterative nature means that already changed code need to be tested again
 - => A lot of **regression testing** needed
 - => Test automation is an important tool in Agile
- Test driven development is often use with Agile



Example of a Kanban Board



Test automation & continuous integration

- Continuous integration
 - Whenever a task is completed, it is integrated to the system
 - Each time the whole set of tests will be executed
 - ⇒ Automated tests are important
- Continuous integration is common in Agile and especially Lean approaches

Summary

- Two ways to improve quality
 - Inspections & reviews
 - Testing
- One big theme was not discussed:
 - Selection of good test cases that will ne discussed in the testing course
- Next lecture (10.3):
 - Quality systems and quality standards
 - General about quality
 - Maybe more about testing