BLG 475E: Software Quality and Testing
Fall 2017 -18

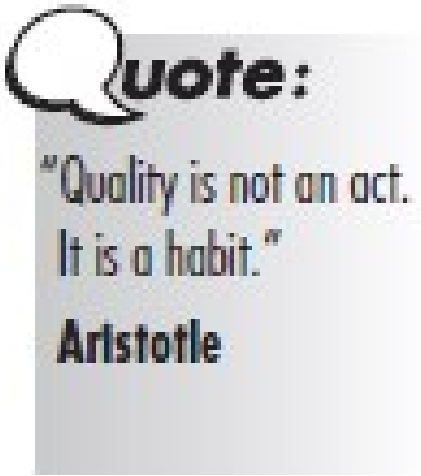
Software defects & peer reviews

Dr. Ayşe Tosun

İstanbul Teknik Üniversitesi

Outline

- Software quality assurance activities
- Classification of defects
- Benefits of defect classification



Software quality assurance activities

- Verification
  - (IEEE, 1990) The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.
- Validation
  - (IEEE, 1990) The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements

Verification and Validation

- Verification
  - (IEEE, 1990) The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.
- Validation
  - (IEEE, 1990) The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements

- Are we building it right?
- Are we building the right product?

A list of VV&T activities

- Code reviews
- Inspections
- Walkthroughs
- Automated prediction models

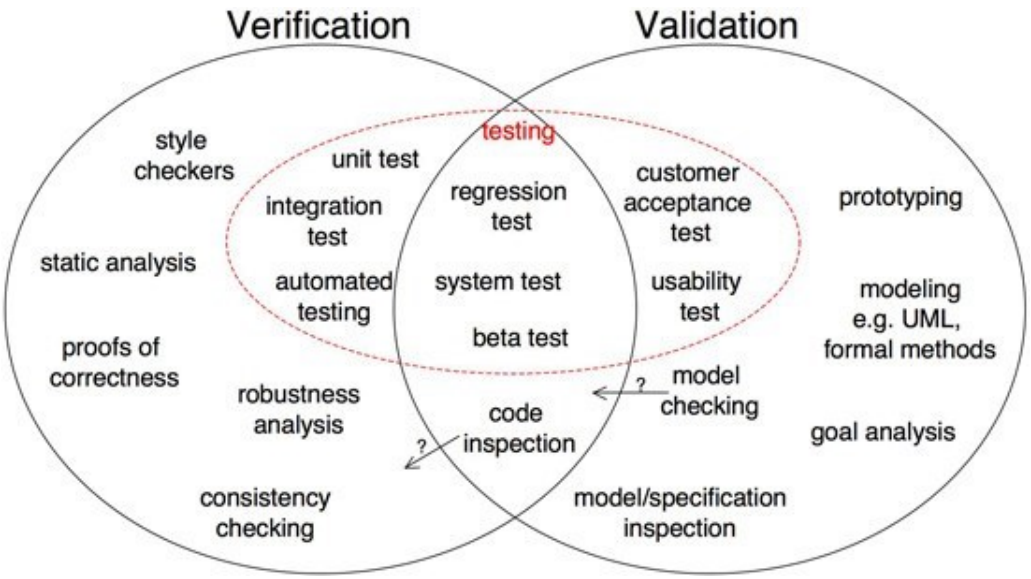
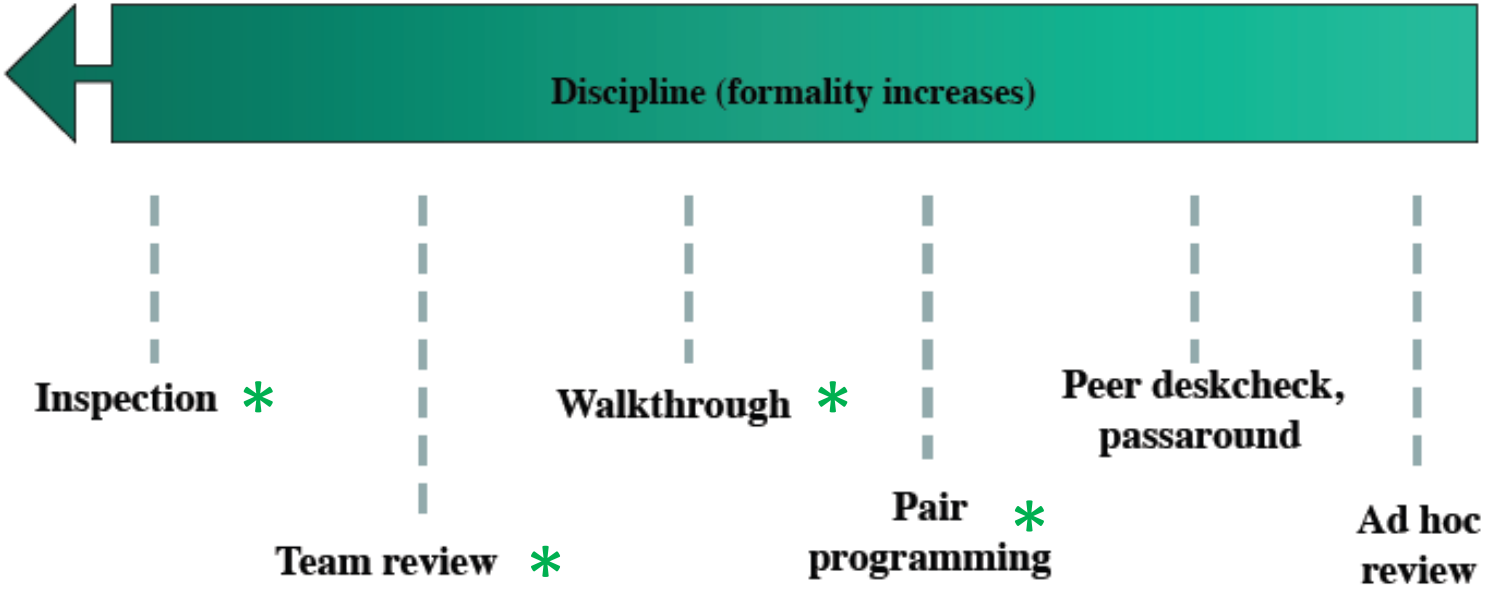


Figure from S. Easterbrook 2010

- Testing

Peer reviews



* We will cover in this lecture.

Based on Wiegers 2002

Peer reviews as a Verification method

- Effective software practitioner should have a rich tool kit of quality techniques (static analyzers, tools that look for run-time problems, test coverage tools).
- These quality tools and practices complement each other.
- Testing cannot demonstrate whether a product complies with standards.
- Different than testing:
  - Reviewers can spot unclear error messages, inadequate comments, hard-coded variables values, unnecessary code.

What can be reviewed?

- According to IEEE Standards for Software Reviews (1999)
  - 37 types of software-related products as candidates for review
  - Source code is often associated with review.

What can be reviewed

- Requirements specification, use cases, analysis models
- Business process models and rules
- Project documents and all kinds of project plans
- Architecture descriptions
- User interface design and prototypes
- Software and database design descriptions and models
- Source code
- Program documentation and system maintenance documentation
- Test plans, designs, cases
- User guides, reference manuals, help screens, tutorials
- Build, release, installation procedures
- Software development procedures, standards

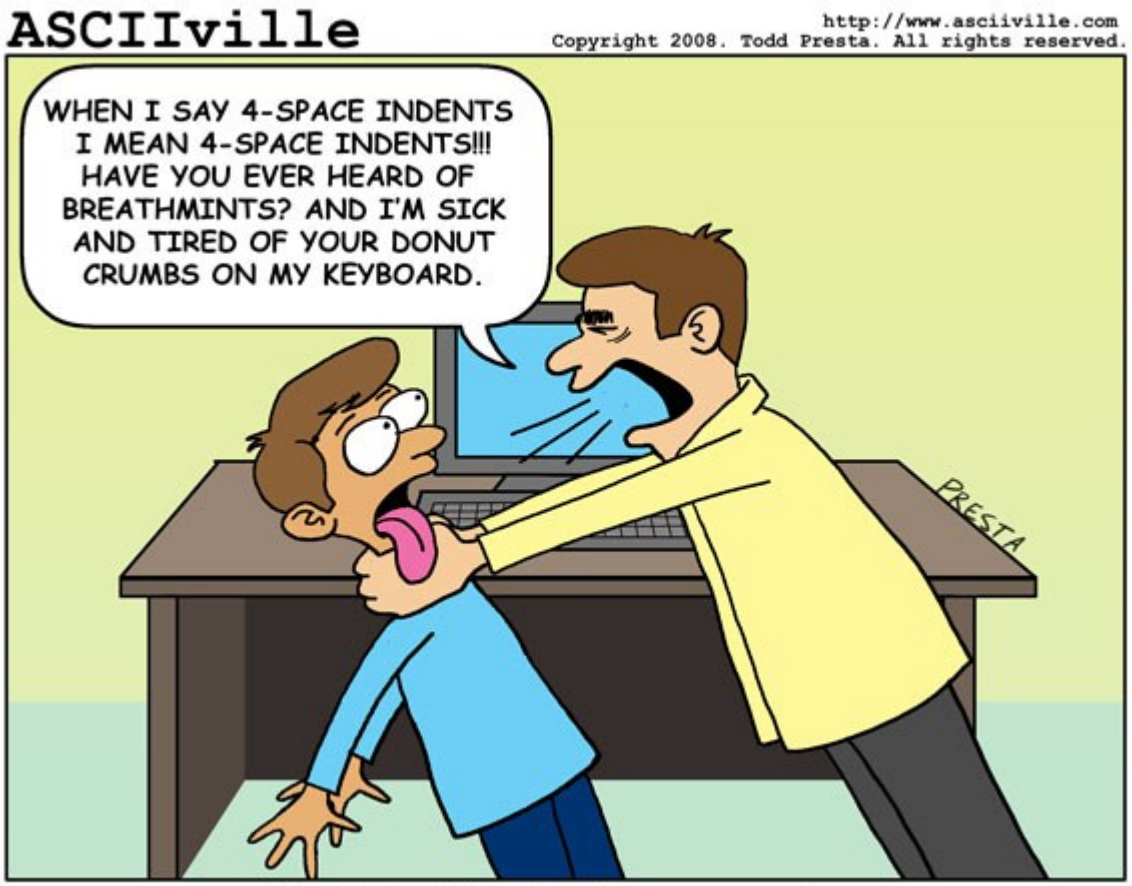
Peer Reviews

- Reviews help build
  - collaborative mindset,
  - with team members willing to share knowledge,
  - learn from each other, and
  - contribute to the quality of each system component.
- Requires involvement of several participants

Activities in Peer Reviews

	Review Activity				
Review Type	**Planning**	**Preparation**	**Meeting**	**Correction**	**Verification**
Inspection	Yes	Yes	Yes	Yes	Yes
Team Review	Yes	Yes	Yes	Yes	No
Walkthrough	Yes	No	Yes	Yes	No
Pair Programming	Yes	No	Continuous	Yes	Yes
Peer Deskcheck, Passaround	No	Yes	Possibly	Yes	No
Ad Hoc Review	No	No	Yes	Yes	No

From Wiegers 2002



The dark side of pair programming.

Pair Programming

- Component of ”agile programming”
- According to Williams and Kessler 2000
  - Promotes collaboration,
  - Attitude of collective ownership of the team’s code base,
  - Shared commitment to the quality of each component

Rules/ Tips in Pair Programming

All production code
... must be developed by a pair.

Both parties contribute to the solution
...switching roles between "driver" and "navigator" frequently.

Change pairs frequently
...three times per day or more

Develop at a comfortable workstation
...that accommodates two people side by side.

End pairing when you get tired
Constrain to no more than ⅔ of your work day.

Source: http://agileinaflash.blogspot.fi/2009/02/abcs-of-pair-programming.html

Walkthroughs

- Informal process
- Author of the work is dominant.
- Other specific review roles are not defined.
- No defined procedure, no exit criteria, no management reporting, no metric generation.
- Analysis by Ford Motor Company (1995)
  - 50% fewer defects per kLOC found compared to inspections.

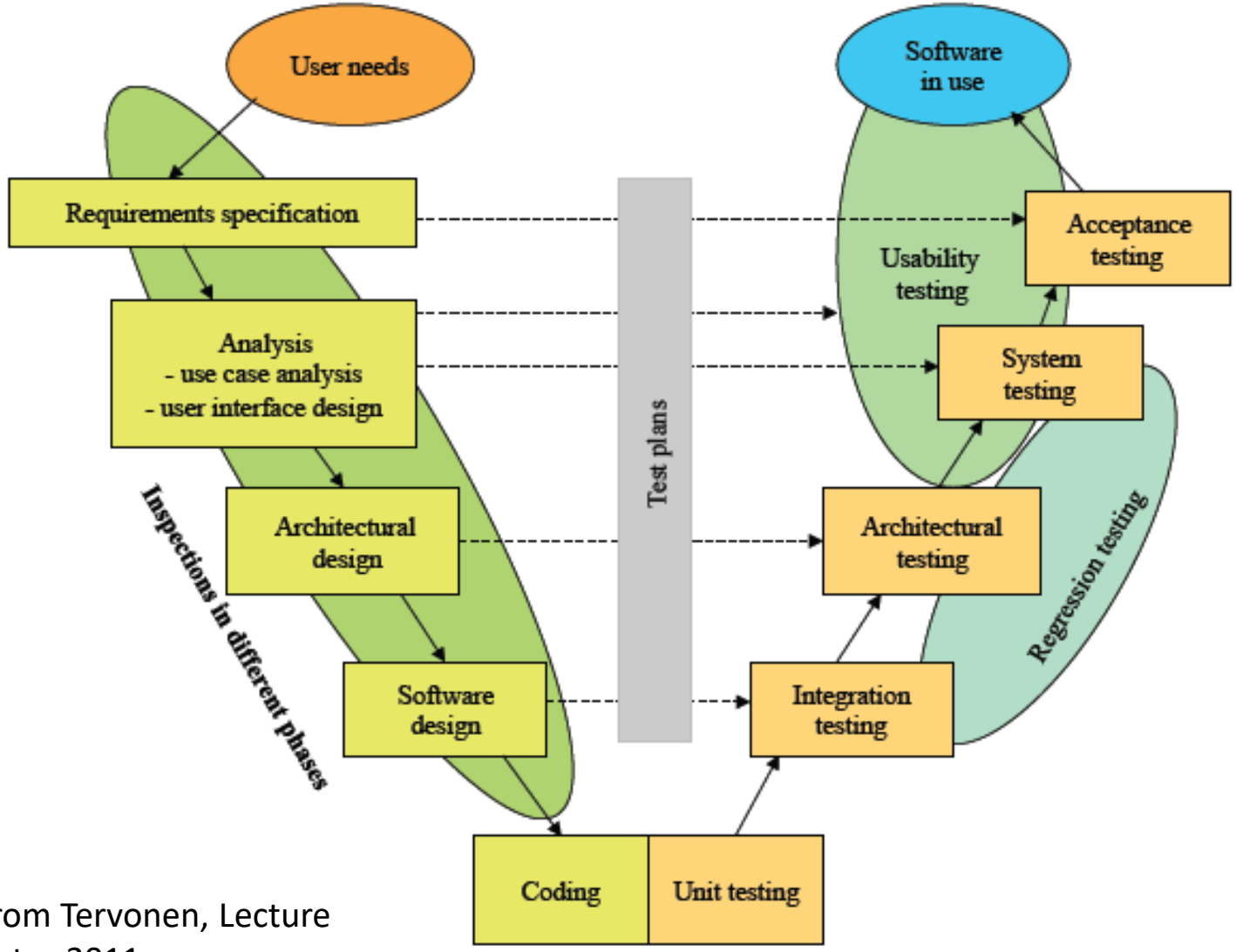
Team Reviews

- ”inspection-lite”
- Group of reviewers
- Cost more than having a single colleague perform a peer deskcheck, but
- Different participants find different problems
- Overview and follow-up are simplified or omitted.
- According to study by Veenendaal 1999
  - Only 2/3 of defects revealed by inspections are found with team reviews.
- IBM Federal Division report 1993
  - ½ of Coding productivity measured in inspections

Inspections

- Most systematic and rigorous type of peer review
- Multistage process with specific roles assigned to participants
- Rely on *checklists* of defects commonly found in different types of software products and other analytical techniques to search for defects.
- Micheal Fagan developed the best known inspection method (1986).

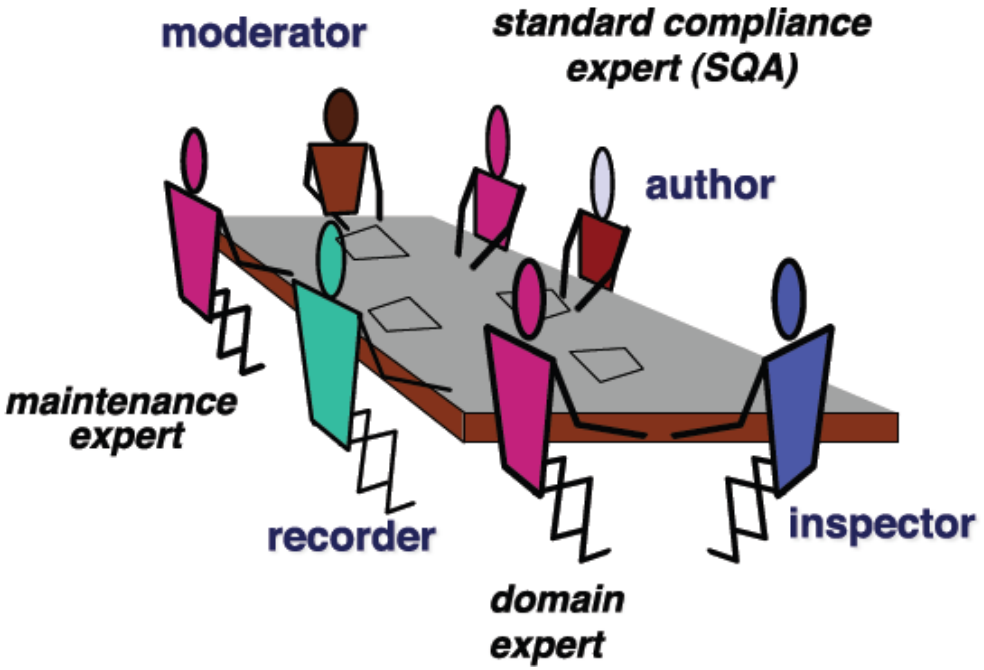
Software inspections

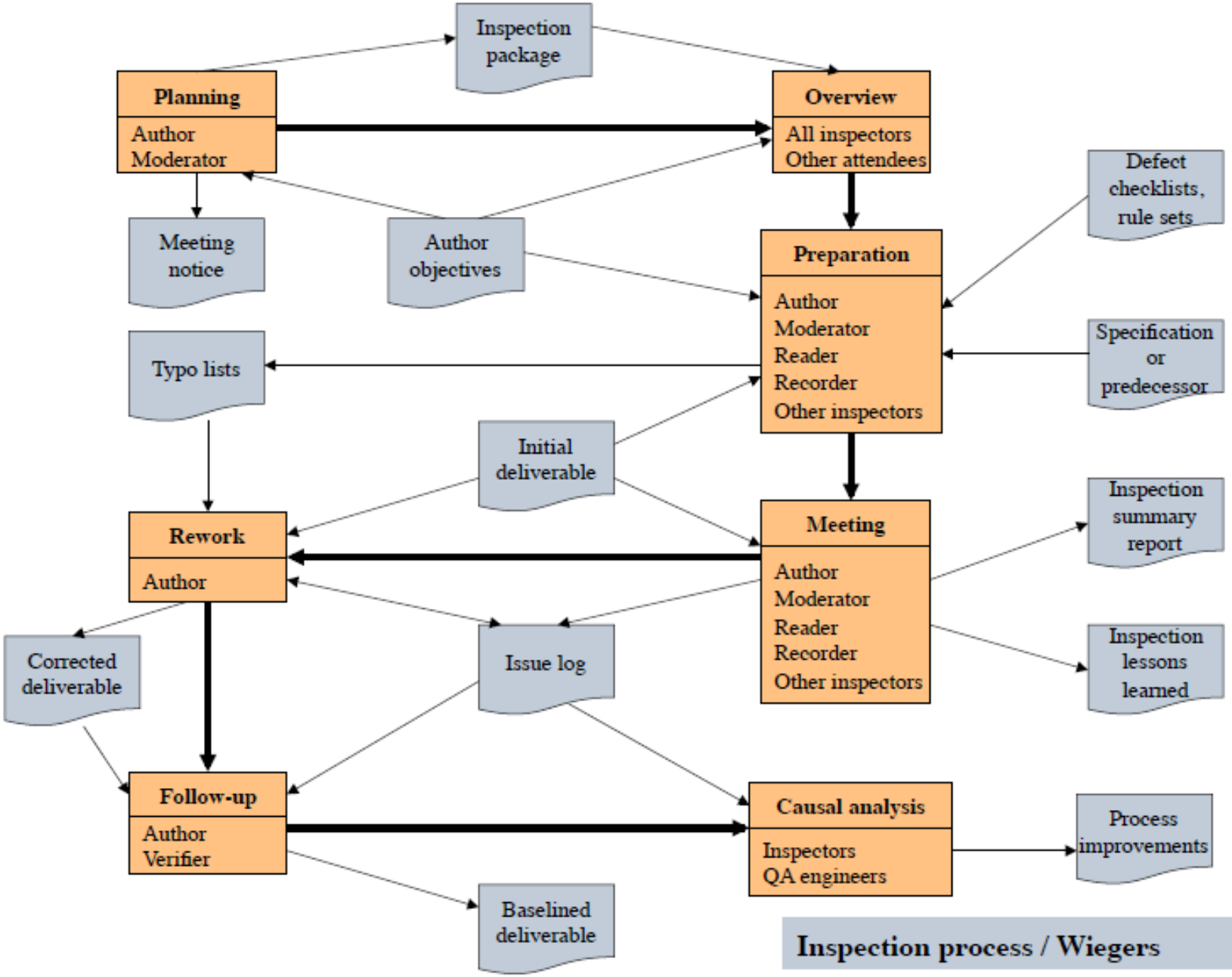


From Tervonen, Lecture notes 2011

Inspection principles

1. Review the product, not the producer.
2. Set an agenda and maintain it.
3. Limit debate and rebuttal.
4. Enunciate (discover) problem areas, but do not spend time for problem solving.
5. Take written notes.
6. Limit number of participants and insist upon advance preparation.
7. Develop a checklist for each product that is likely to be reviewed.
8. Allocate time and resources.
9. Conduct meaningful training for all reviewers.
10. Review your early reviews.





Inspection process / Wiegers

Wiegers 2002

Inspection Issue Log

Project: _____	**Origin:**	Requirements, Design, Implementation, Testing
Inspection ID: ____	**Type:**	Missing, Wrong, Extra, Usability, Performance, Style, Clarity, Question
Meeting Date: **Recorder:**	**Severity:**	Major, minor

Defects Found: ____ Major, ____ minor **Defects Corrected: ____ Major ____ minor**

	Origin		Type		Severity		Location		Description
1.									
2.									
3.									
4.									
5.									
6.									

Inspection Summary Report

Inspection Identification:

Project: _____
Inspection ID: _____
Meeting Date: _____

Work Product Description:

	Inspectors	Signature	Preparation Time
Author:	_____	_____	_____ hours
Moderator:	_____	_____	_____ hours
Recorder:	_____	_____	_____ hours
Reader:	_____	_____	_____ hours
Inspector:	_____	_____	_____ hours

Inspection Data

❑ Pages or ❑ Lines of Code:
Planned for Inspection: _____
Actually Inspected: _____

Meeting Time: _____ hours
Total Planning Effort: _____ labor hours
Total Overview Effort: _____ labor hours
Total Preparation Effort: _____ labor hours
Actual Rework Effort: _____ labor hours

Product Appraisal

ACCEPTED
___ as is
___ conditionally upon verification
Verifier: _____

NOT ACCEPTED
___ reinspect following rework
___ inspection not completed

Projected Rework Completion Date: ____

Choosing a Review Method (from Wiegers 2002)

Review Objectives	Inspection	Team Review	Walkthrough	Pair Programming
Find implementation defects	X	X	X	X
Check conformance to specifications	X	X		
Check conformance to standards	X			
Verify product completeness and correctness	X		X	
Assess understandability and maintainability	X	X		X
Demonstrate quality of critical or high-risk components	X			
Collect data for process improvement	X	X		
Measure document quality	X			
Educate other team members about the product		X	X	
Reach consensus on an approach		X	X	X
Explore alternative approaches			X	X
Ensure that changes or bug fixes were made correctly		X	X	
Simulate execution of a program			X	
Minimize review cost				
Provide progress check to management and customers				

In-class exercise for inspections

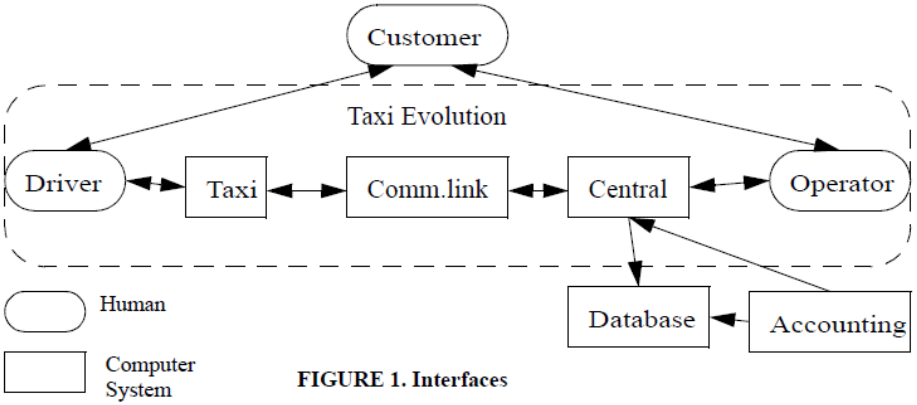


FIGURE 1. Interfaces

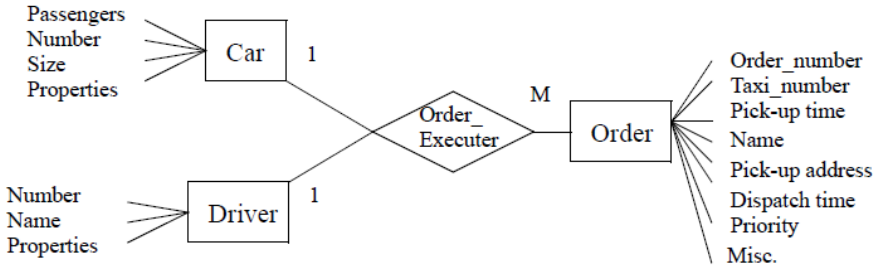


FIGURE 2. Database design

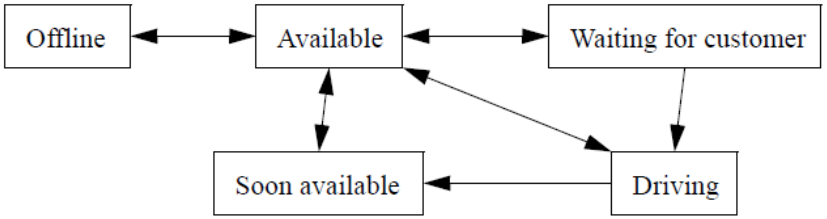


FIGURE 3. States of the taxi

In-class exercise round 1

- Read the guidelines and use cases of the taxi system provided to you.

1. Inspect the document **individually**.

2. Log the time and faults you found in the document.

3. In the logs, include the faults' locations in the document.

In-class exercise round 2

- Divide in groups of four.
- Compile the faults you have found by identifying the common faults found, the faults that only one of you found and false positives.
- Roles:
  - Moderator
  - Inspector
  - Quality assurance expert (reader-keeps notes and submits the report at the end of the lecture)
  - Architect

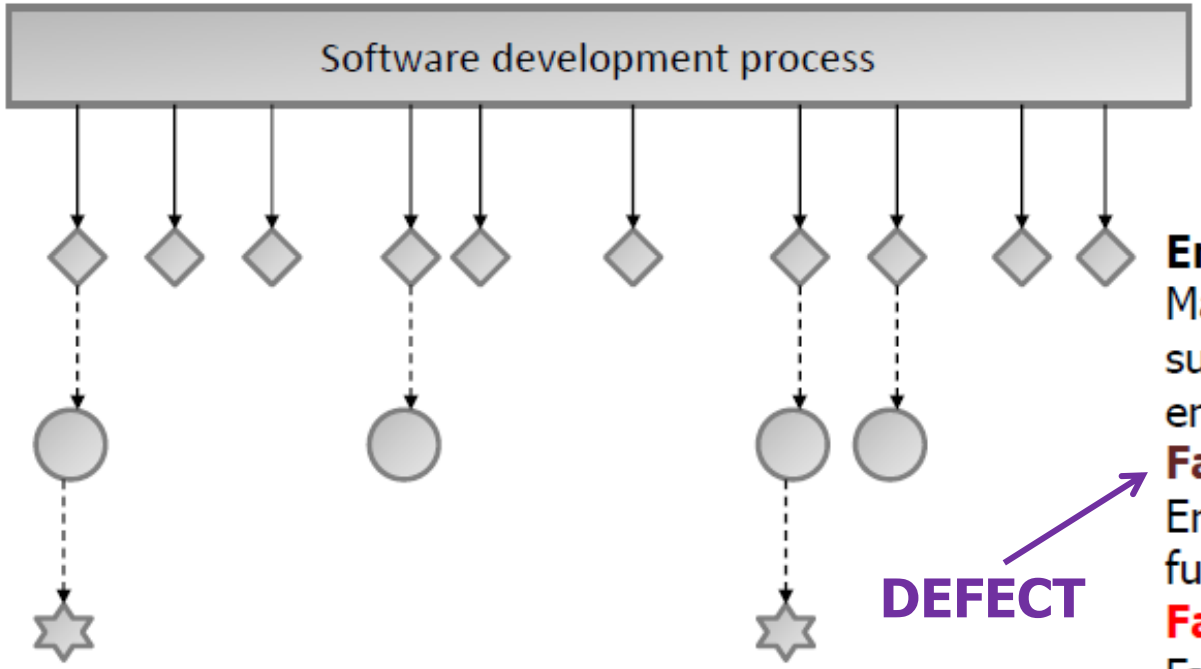
Defect estimation with a Capture-Recapture method (in Inspections)

If two inspectors are assigned to inspect the same product

- d1: defects found by Inspector 1
- d2: defects found by Inspector 2
- d12: defects detected by Inspector 1 and Inspector 2
- Nt: total defects (detected and undetected)
  - Nt: (d1*d2)/d12
- Nr: remaning defects (undetected)
  - Nr: Nt – (d1 + d2 – d12)

Software defects

Remember from the previous lectures



Error
Made by programmer such as grammatical error or logical error

Fault
Error can cause improper functioning of a software

Failure
Fault becomes failure only when it is "activated".

DEFECT

Figure from (Galin, 2004)

What does "Bug" correspond to?

Extremes of the Spectrum

Statistical defect models

vs.

Qualitative causal models

Reliability growth models

- # residual defects
- failure rate
- short term defect detection rate

ODC to fill the gap between

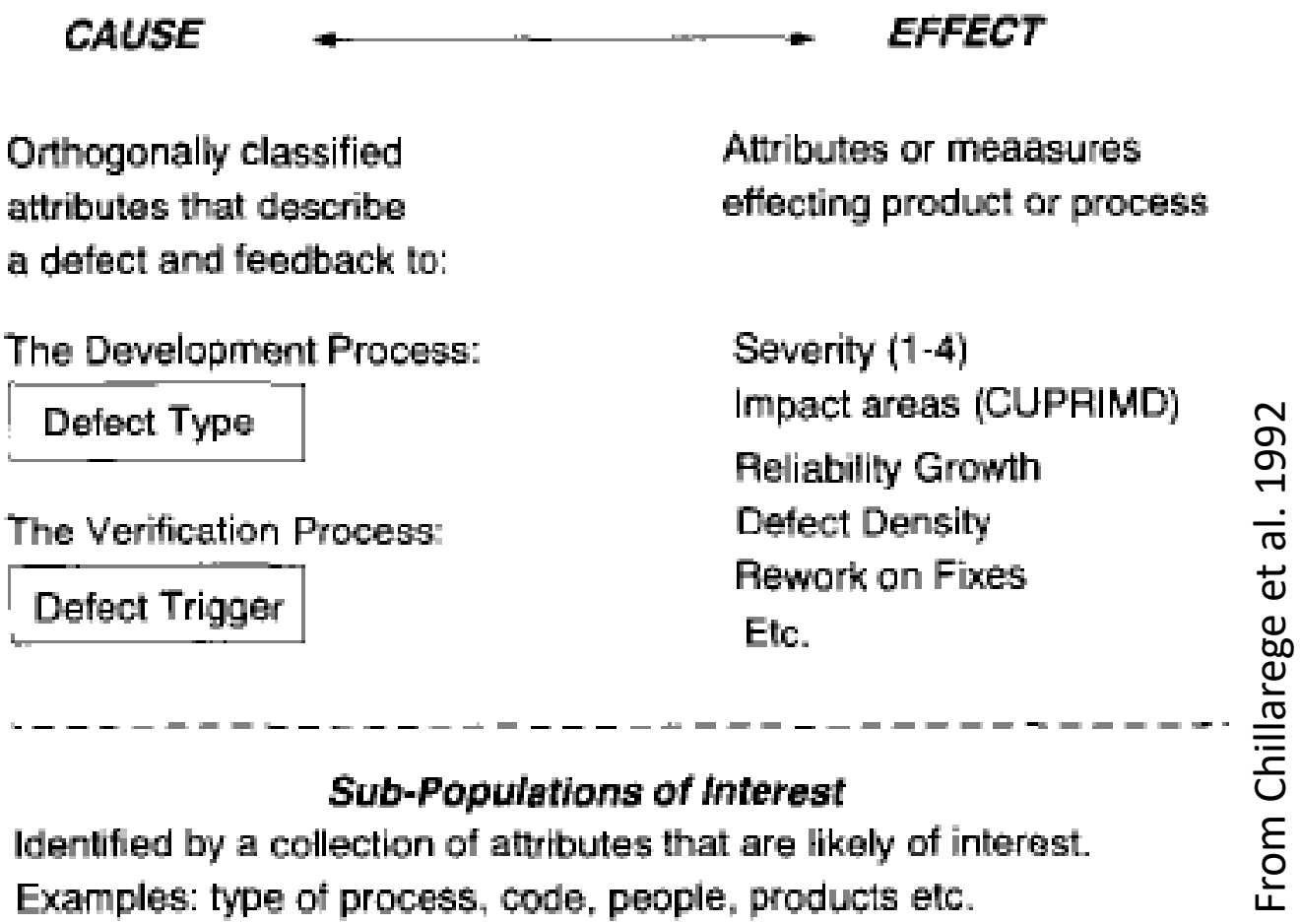
Causal models

- Root causes of failures
- Qualitative analysis
- Provide feedback to developers

Orthogonal Defect Classification (ODC)

- Categorization of defects into classes that collectively point to the part of the process that needs attention
  - Classification scheme should have consistency between stages.
  - Classification should be independent of the specifics of a product or organization. (orthogonality)
  - Classification should be simple and obvious to a programmer, without much room for confusion.

ODC data to build cause-effect relations



From Chillarege et al. 1992

”Defect Type” Attribute

- Programmer chooses the type.
- Missing/incomplete or incorrect

Defect Type	Missing or Incorrect	*Process Associations*
Function	Select One	DESIGN
Interface		LOW LEVEL DESIGN
Checking		LLD or CODE
Assignment		CODE
Timing/Serilization		LOW LEVEL DESIGN
Build/Package/Merge		LIBRARY TOOLS
Documentation		PUBLICATIONS
Algorithm		LOW LEVEL DESIGN

From Chillarege et al. 1992

ODC Defect types

- Function
  - affects significant capability, end-user interfaces, product interfaces, interface with hardware architecture, or global data structure(s) and should require a formal design change.
- Interface
  - errors in interacting with other components, modules or device drivers

ODC Defect types

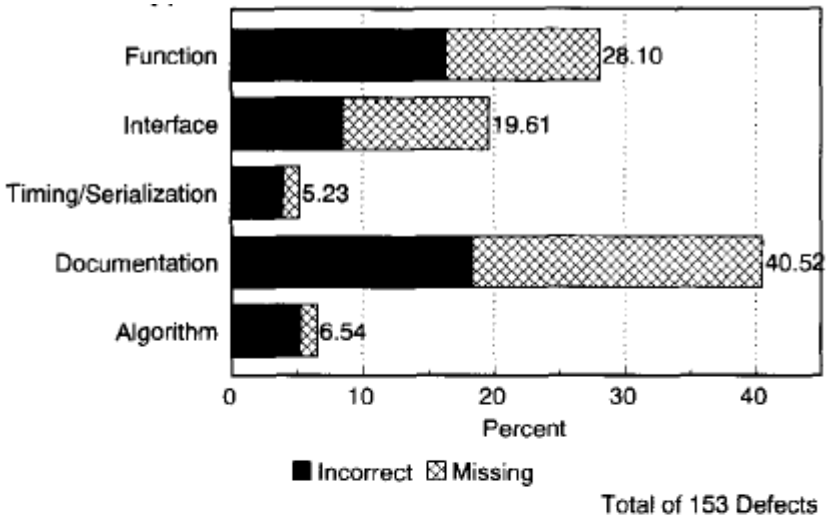
- Assignment
  - value(s) assigned incorrectly or not assigned at all.
- Timing/Serialization
  - necessary serialization of shared resource was missing, the wrong resource was serialized,
  - or the wrong serialization technique was employed
  - wrong implementation in the sequence of using resources
- Build/Package/Merge
  - occur due to mistakes in library systems, management of changes, or version control

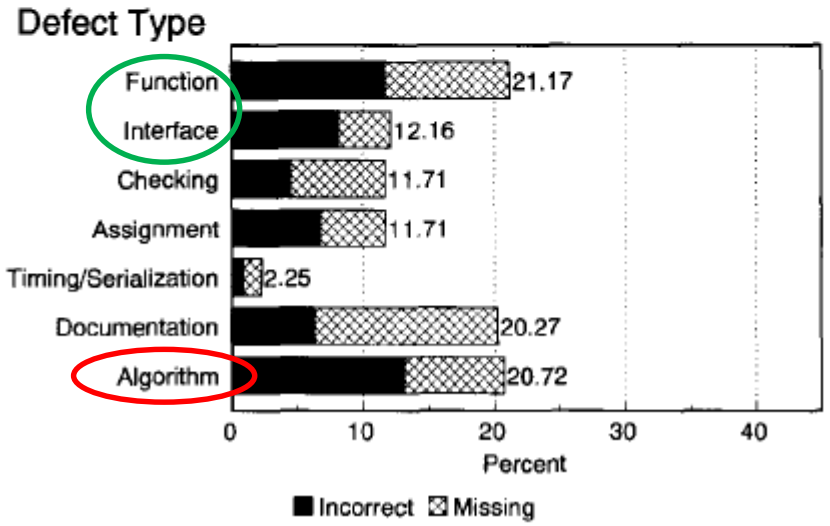
ODC Defect types

- Documentation
  - affect publications and maintenance notes
- Checking
  - missing or incorrect validation of parameters or data in conditional statements
- Algorithm
  - include efficiency or correctness problems that affect the task and can be fixed by (re)implementing an algorithm or
  - local data structure without the need for requesting a design change

Change in defect types

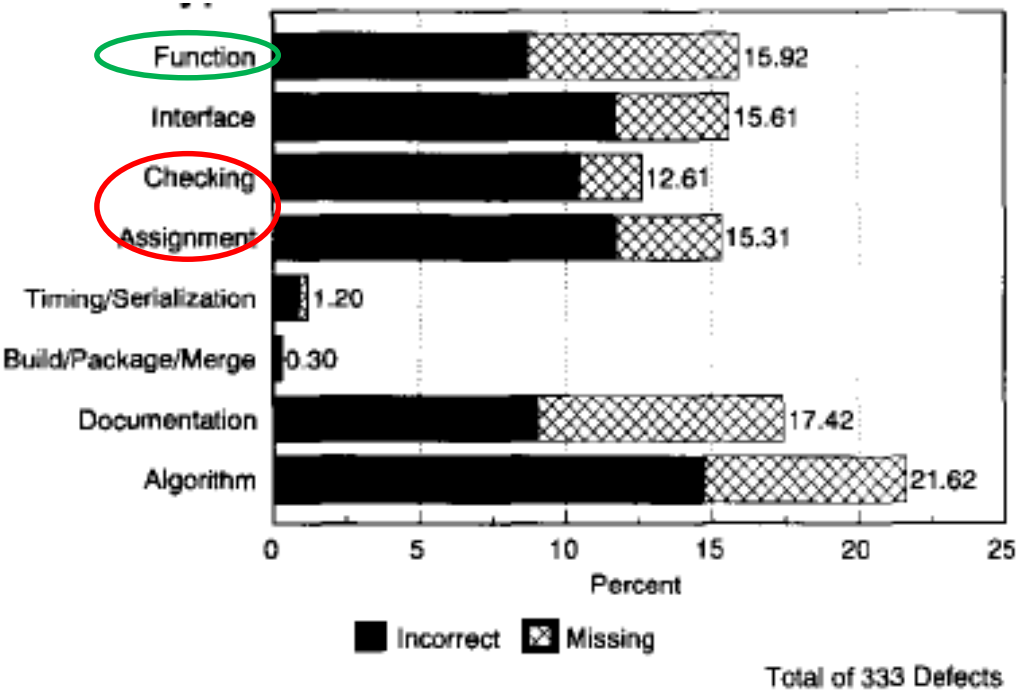
- Which chart corresponds to finding defects during
  - Low level design?
  - High level design?





Figures from Chillarege et al. 1992

Change in defect types

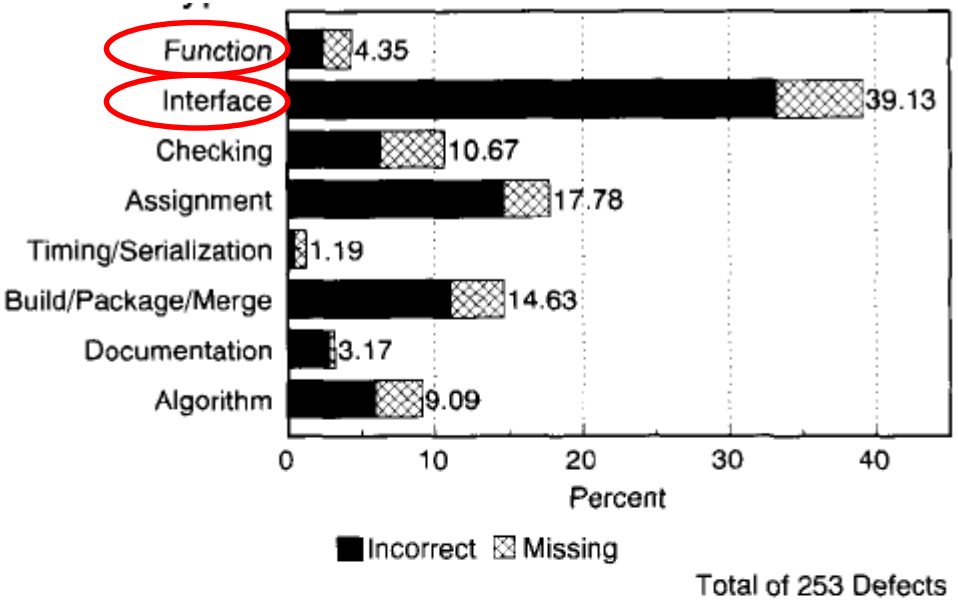


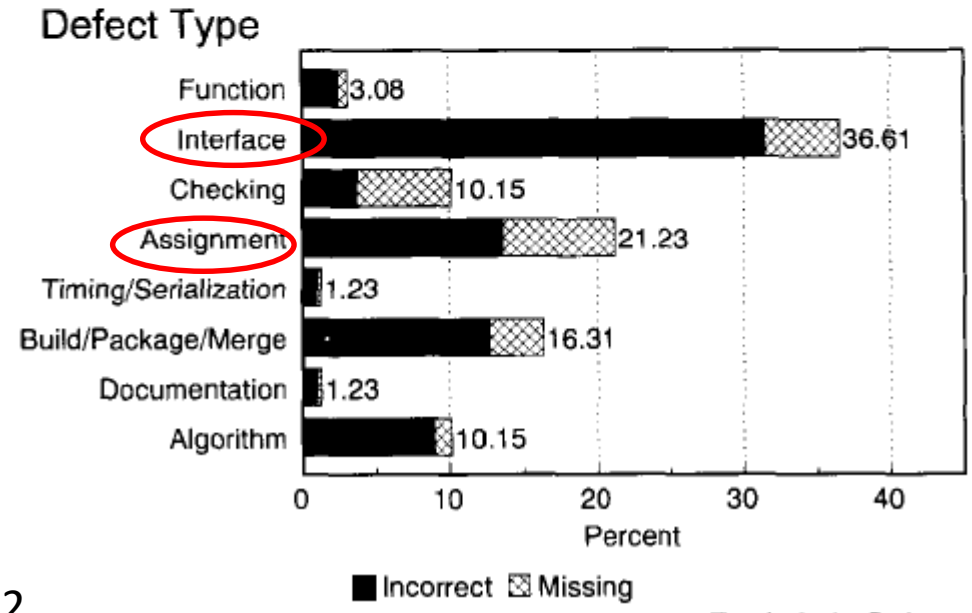
In Code Inspection

Figures from Chillarege et al. 1992

Unhealthy trends on defect types

- Top: Function verification
- Bottom: System test





Figures from Chillarege et al. 1992

General quality evaluation models

A segmented model for reliability level estimation

Product type	Failure rate (per hour)	Reliability level
Safety-critical software	$< 10^{-7}$	Ultra-high
Commercial software	10^{-3} to 10^{-7}	Moderate to high
Auxiliary software	$> 10^{-3}$	Low

Product specific models

Defect distribution for previous releases of a product

Requirement	Design	Coding	Testing	Support
5%	10%	35%	40%	10%

e.g. if you catch 20 design defects → 20 / 10% = 200 predicted defects in total

From Tian 2004

Distribution of defects & cost of defect removal

No.	Software development phase	Average percentage of defects originating in phase
1	Requirements specification	15%
2	Design	35%
3	Coding (coding 30%, integration 10%)	40%
4	Documentation	10%

Representative ratios based on Boehm, 1981 and Pressman 2000

No.	Software development phase	Average relative defect cost (cost units)
1	Requirements specification	1
2	Design	2.5
3	Unit tests	6.5
4	Integration tests	16
5	System tests / acceptance tests / system documentation review	40
6	Operation by customer (after release)	110

Figures from Galin, 2004

Benefits of defect classification

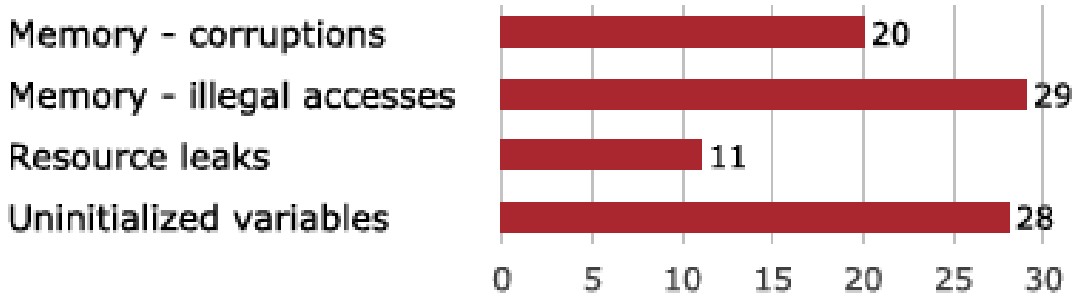
- Example
  - Coverity 2010 Report on Open Source Evolution
- What is Coverity?
  - Development testing platform operates as a safety net for developers, enabling them to deliver innovation quickly while protecting the business from the introduction of critical defects
    - Different modules (Static analysis, Dynamic analysis, Integration with FindBugs, Architecture analysis, etc.)

Coverity Report for Android Kernel

- Analysis on Android kernel 2.6.32
  - Targeted for smartphones with additional support for wireless, touchscreen, and camera drivers.
- Conclusions
  - The Android kernel used in the HTC Droid Incredible has approximately half the defects that would be expected for average software of the same size. (Integrity Level 1)
  - Android-specific code that differs from the Linux kernel had about twice the defect density of the core Linux kernel components.
  - The Android kernel has better than industry average defect density (one defect for every 1,000 lines of code).
  - Quote: 'We found 88 high-risk and 271 medium-risk defects in Android.'

High-Risk Defects

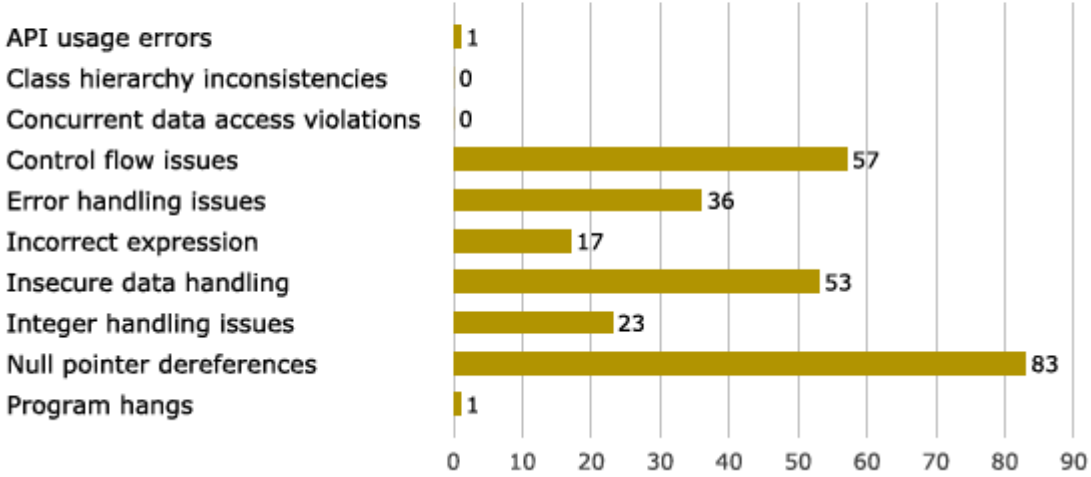
High-impact defects that cause crashes, program instability, and performance problems.



Android Kernel report from Coverity, 2010.

Medium-Risk Defects

Medium-impact defects that cause incorrect results, concurrency problems, and system freezes.



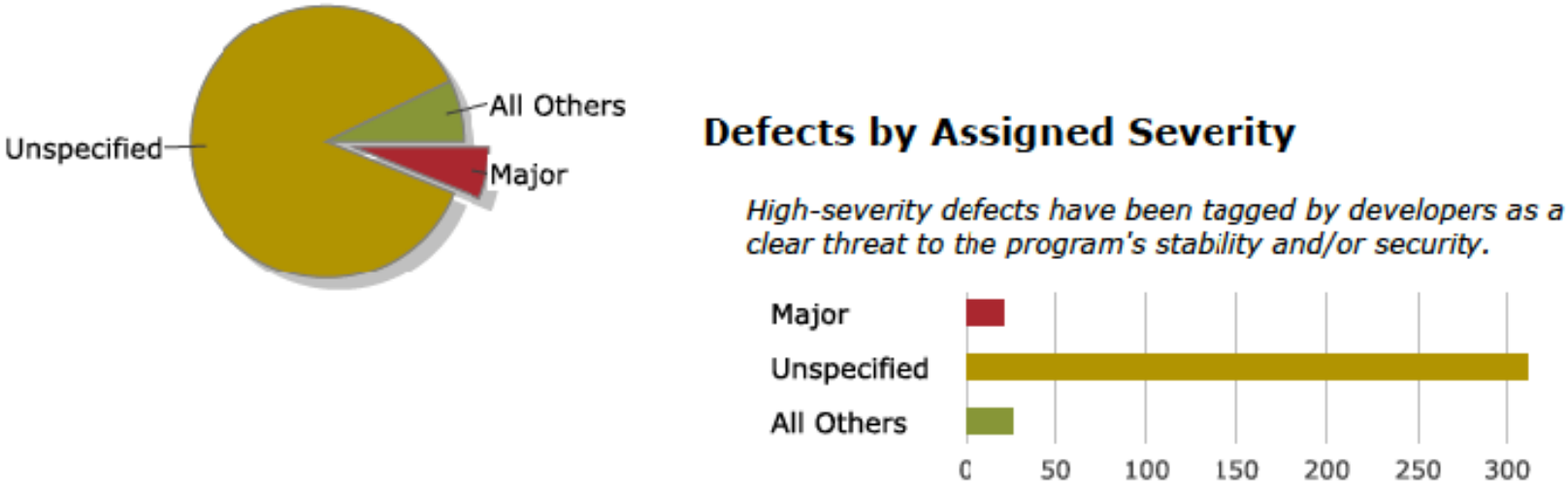


Figure 5. Android kernel defects by assigned severity.

Android Kernel report from Coverity, 2010.

References

- Chillarege, I.S. Bhandari, J.K. Chaar, M.J. Halliday, D.S. Moebus, B.K. Ray, M.Y. Wong. 1992. Orthogonal Defect Classification – A Concept for In Process Measurements, IEEE Transactions on Software Engineering, (18:11), pp. 943-956
- Wiegers K.E., Peer Reviews in Software, Addison-Wesley, 2002.
- Galin D., Software Quality Assurance: From theory to implementation, Addison Wesley, 2004.
- Tian J., Quality-Evaluation Models and Measurement, IEEE Software, May/June 2004
- Coverity Inc. 2010. Scan Report and Android Integrity Report, http://softwareintegrity.coverity.com/2011ScanAndroidReg.html