QUATIC 2014

Software defects:

Stay Away from them.

Do Inspections!

Guilherme Horta Travassos
Universidade Federal do Rio de Janeiro
COPPE/PESC
CNPq Researcher, **ISERN** Member
ght@cos.ufrj.br
www.cos.ufrj.br/~ght

Agenda

- Software Systems
  - Characteristics
  - Software Engineers Reality
- Software Systems Development Issues
- Software Defects
- Inspection Method and Techniques
- Evidence on Software Inspections (academia and industry)
- Conclusion

Software Systems

used largely by people other than s… developers

users may be from differe… kground, so a proper user interface n… provided

portability is key

It must be t…ghly verified, validated and tested before i… rational use

Software is everywhere…

Software Systems

Early years
Custom Software
Standalone
Batch

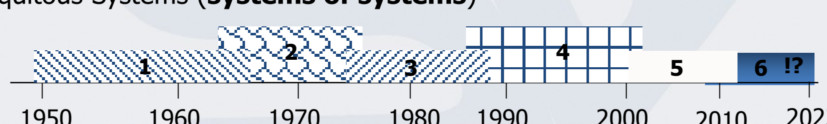
Second Stage
Multi-user
Real-time
Database
Product Software

Third Stage
Distributed Systems
Embedded "intelligence"
Low cost hardware
Consumer Impact

Fourth Stage
Powerful desk-top systems
Object-oriented technologies
Expert systems
Artificial neural networks
Parallel computing
Network computers

Fifth Stage
Multi-skilled, geographically distributed development
Componentry (reuse and recycling)
Development and evolution models, including biological analogies
Interdependence among design, business, and evaluation
Agile software manufacture
Empowering the domain expert (vs. maintaining integrity)
Non-scripting development languages

Sixth Stage
"mobile" apps
Large Scale Science (e-science) with intensive use of e-infrastructure
Ubiquitous Systems (**systems of systems**)

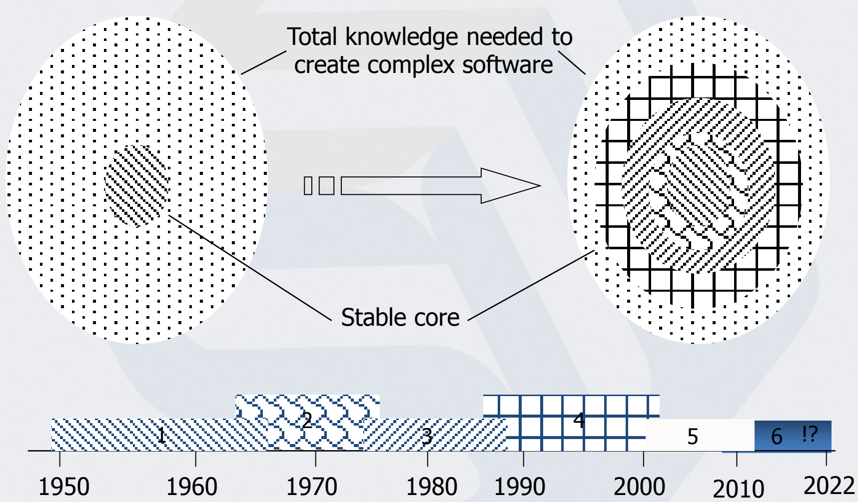


Adapted from PRESSMAN, R. S. , 1994, "*Software Engineering: A Practitioner's Approach*", *European Edition, McGraw-Hill.*

Software Systems

System Software
Real-Time Software
Business Software
Embedded Software
Engineering and Scientific Software
Personal Computer Software
Artificial Intelligence Software
Ubiquitous Software
Mobile Apps
Systems of Systems

Software Systems



Adapted from PRESSMAN, R. S. , 1994, "*Software Engineering: A Practitioner's Approach*", *European Edition, McGraw-Hill.*

Some Software Systems Characteristics

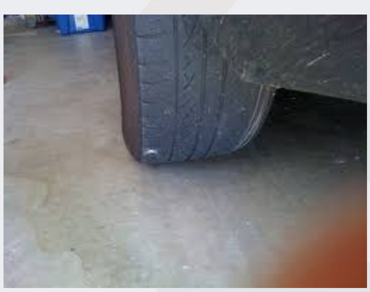
Software can not be manufactured (in the classical sense)

X

<u>Software costs concentrate in engineering</u>

Some Software Systems Characteristics

Software doesn't "wear out", but it deteriorates



Hardware

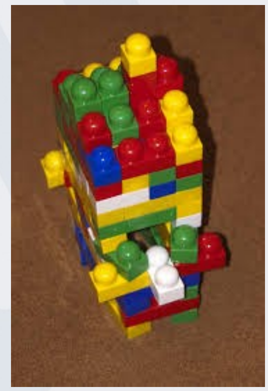
X

Software

Some Software Systems Characteristics

Custom-built rather than assembled from existing (quality) components

X



Some Software Systems Characteristics

Computers everywhere demand softw... ve made society increasingly dependent ... bility systems.

Enormous economic damage and potential human suffering can occur when software systems fail

Hardwa... ...nts continue to outpace our ability to build software ... hardware's potential

Software Engineers Reality...

All software systems fail...

All software systems fail...

- A full list of evidence at http://catless.ncl.ac.uk/Risks/
  - **John Oates, Who's to blame this time? *The Register*, ...** "The London Stock Exchange has suffered yet another systems ... high and dry since 9.30 this morning. The Exchange last wen... and took almost the entire day to get back online. T... Exchange's busiest days, was the day after the $... Freddie Mac and Fannie Mae, leading to lots ... operation at 14.00.]"
  - **Hacking ring steals $9 million fr...** ... FBI press release, a global ring of hackers ... dit processing company, stole PIN number... ed to steal 9 million USD from over 2000 AT... st been brought to justice.)"
  - **Total Parent...** ... tal parenteral nutrition (intravenou... er and there are tools to assist in the preparation ... such nutrition is typically administered weeks to years ... s to change frequently (in instances, daily) and because pa... t of treatment are invariably quite ill, even relatively small fla... ions can produce significant physiological disturbances."

more recently
US VISA System,
Citius Platform,...

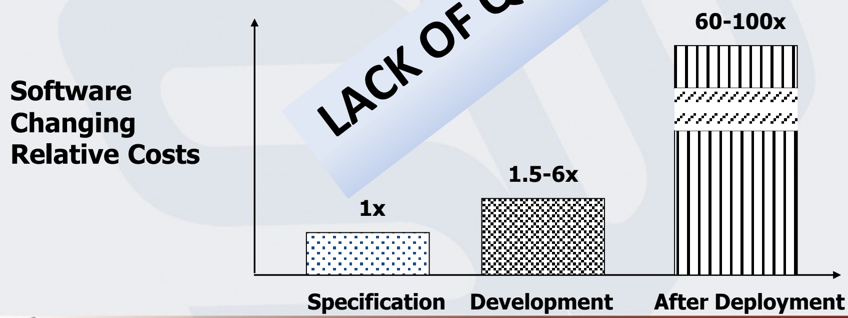
. . .

Software Systems: related persistent problems

We struggle to build high reliability and quality software

<u>However, our ability to support and enhance existing so</u>ftware is still
<u>threatened by poor design and insufficient resourc</u>

LACK OF QUALITY



Software Engineers Reality...

All softv ems fail...

WHY?

Software Engineers Reality...

Software systems construction does not follow a smooth pathway...

Software Systems Construction

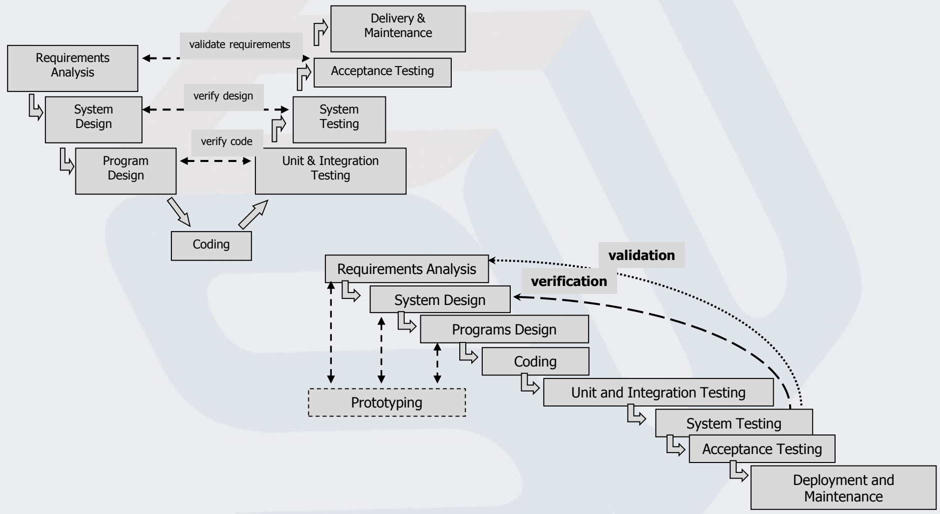
In general, it follows a Software Development Process specifying:

the adopted software life-cycle and paradigm
the software technologies (methods, tools) to be used throughout the development time
who participates (roles) and when
the management, quality and verification, validation and testing plans

It defines how multiple developers can communicate and cooperate

Software Systems Construction

Some software life-cycle shapes



Software Engineers Reality...

Software De… …ocesses
demand s… …ologies, but...

Not enough evidence regarding software technologies...

Some Software Technologies Pitfalls...

As it has been recently commented by Forrest Shull (Keynote at ICGSE, 2012):

Requirements Elicitation: 30 studies covering 43 different techniques over 20 years of research

Dieste, O., Juristo, N., and Shull, F. "Understanding the Customer: What Do We Know about Requirements Elicitation?" IEEE Software, vol. 25, no. 2, pp. 11-13, March/April 2008.

SW Process Capability/Maturity Models: 61 studies; 52 process models.

von Wangenheim, C., Hauck, J., Zoucas, A., Salviano, C., McCaffery, F., and Shull, F. "Creating Software Process Capability / Maturity Models," IEEE Software, vol. 27, no. 4, pp. 92-94, July / August 2010.

Distributed SW Development: "Few of the models from our review were evaluated…"

Prikladnicki, R., Audy, J. L. N., and Shull, F. "Patterns in Effective Distributed Software Development," IEEE Software, vol. 27, no. 2, pp. 12-15, March / April 2010.

SPL Testing Techniques: 60% of papers describe "solutions or conceptual proposals," while "just a few" report experiences from real development environments.

da Mota Silveira Neto, P.A.; Runeson, P.; do Carmo Machado, I.; de Almeida, E.S.; de Lemos Meira, S.R.; Engstrom, E.; , "Testing Software Product Lines," Software, IEEE , vol.28, no.5, pp.16-20, Sept.-Oct. 2011.

Some Software Technologies Pitfalls...

And also observed in some of our investigations:

Cost Estimation Models: 11 studies (including 2 replications) using different data sets. No evidence about feasibility of models nor possibility of aggregation

Kitchenham, B. ; Mendes, E. ; Travassos, G. H. (2007). Cross versus within-company cost estimation studies: A systematic review. IEEE Transactions on Software Engineering, v. 33, p. 316-329, 2007. http://dx.doi.org/10.1109/TSE.2007.1001

Model based Testing: from 85 selected papers (representing 71 approaches), 27% are speculative, 45% just present simple using examples, 15% show proof of concepts, 5% report some experience and 8% have been experimented.

Dias Neto, A. C. ; Subramanyan, R. ; Vieira, M. E. R. ; Travassos, G. H. ; Shull, F. .(2008) Improving evidence about software technologies: A look at model-based testing. IEEE Software, v. 25, p. 10-13, 2008. http://dx.doi.org/10.1109/MS.2008.64

Testing Stop Criteria: 74 criteria (3 repeated) resulting in 108 variations. Most of them regard software reliability. Others are specific. Just 27% have been evaluated, without evidence about their feasibility (no context indication)

Vidigal, V., Travassos, G. H. (2013). A quasi -systematic review on Testing Stop Criteria. WAMPS 2013.

Some Software Technologies Pitfalls...

And also observed in some of our investigations:

Agility Characteristics and Agile Practices: More relevant characteristics to introduce agility in software processes are concerned with communication, understandability and adaptation (not with agile methods). The agile practices Presence of Client and Planning Poker are not relevant. However, Continuous Integration and Backlog are highly relevant.

De Mello, R.M.; Silva, P.C.; Travassos, G.H. (2014). Agility in Software Processes: Evidence on Agility Characteristics and Agile Practices. SBQS 2014.

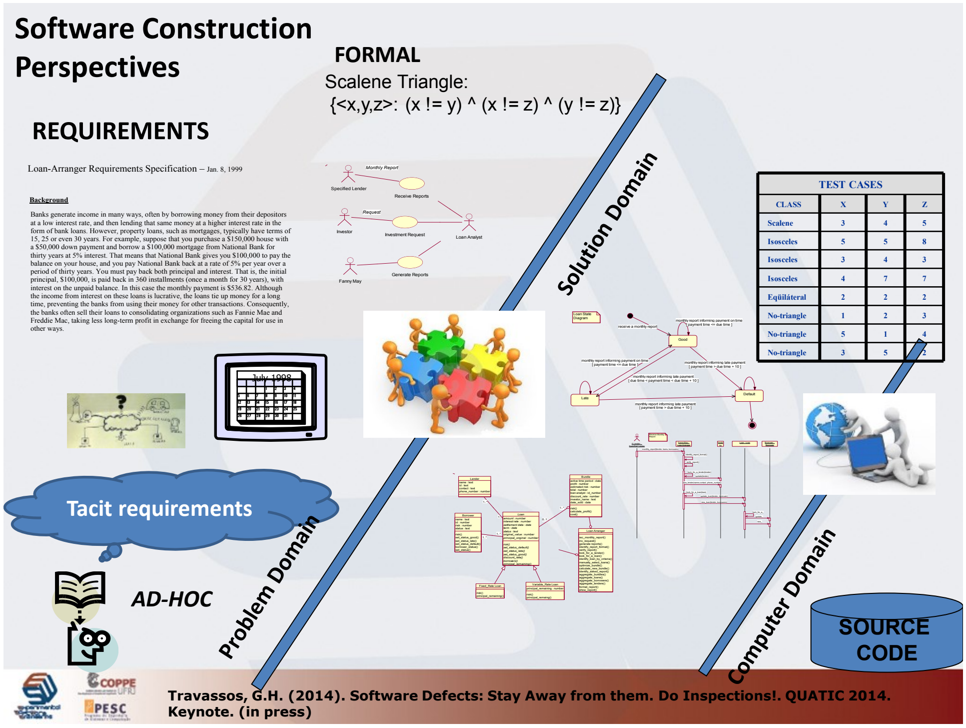
Estimation of Software Testing Effort: There is no consensus about software testing and what can be considered effort regarding it. Therefore, current models and factors are not generically adequate and to use one or another model is risky.

Souza, T.S.; Ribeiro, V. V.; Travassos, G.H. (2014). Software Testing Estimation Effort: Models, Factors and Uncertainties. CACIC 2014 (in press)

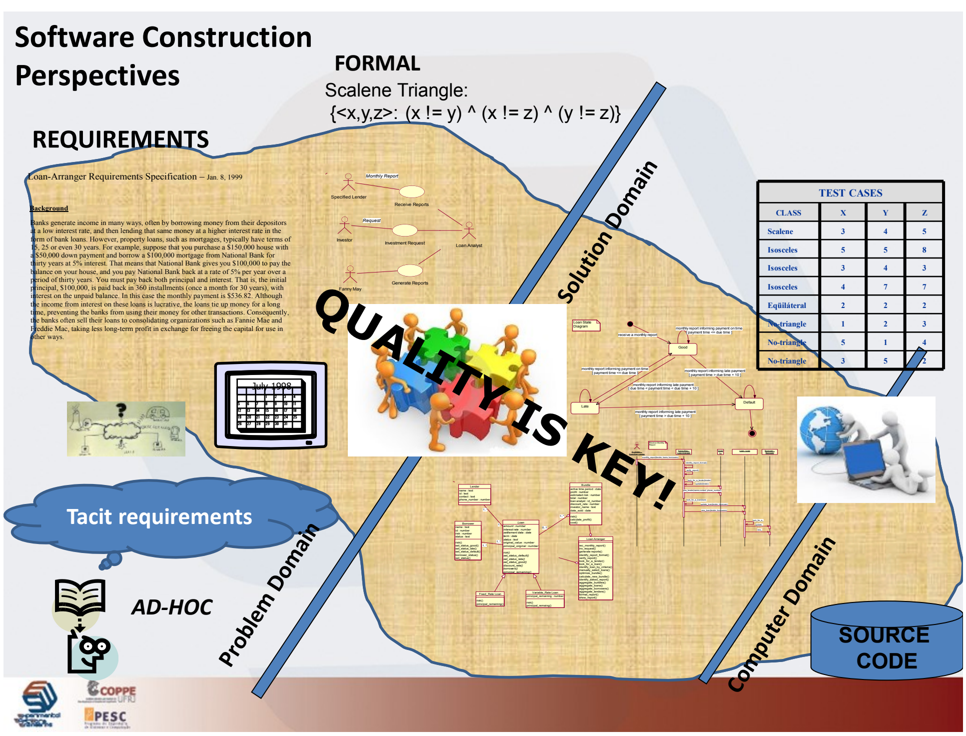
Software Engineers Reality...

Software Develop ...s require communicati ...ration among deve ...keholders...

not easy to guarantee communication and collaboration...



CLASS	X	Y	Z
Scalene	3	4	5
Isosceles	5	5	8
Isosceles	3	4	3
Isosceles	4	7	7
Eqüiláteral	2	2	2
No-triangle	1	2	3
No-triangle	5	1	4
No-triangle	3	5	2



CLASS	X	Y	Z
Scalene	3	4	5
Isosceles	5	5	8
Isosceles	3	4	3
Isosceles	4	7	7
Eqüiláteral	2	2	2
No-triangle	1	2	3
No-triangle	5	1	4
No-triangle	3	5	2

Software Engineers Reality...

Lack of Quality, due…

Software Defects

Software Defect

Error: *a human action that produces an incorrect result*.

Fault: *a manifestation of an error in software*.

Failure: ***(a)*** *termination of the ability of a product to perform a required function or its inability to perform within previously specified limits; or* ***(b)*** *an event in which a system or system component does not perform a required function within specified limits*.

Defect:

an imperfection or deficiency in a work product where that work product does not meet its requirements or specifications and needs to be either repaired or replaced.

It is a fault when detected during the execution of software

IEEE Std. 1044-2009. (2010). Classification for Software Anomalies.

Software Defects

Most of them results from human based activities!

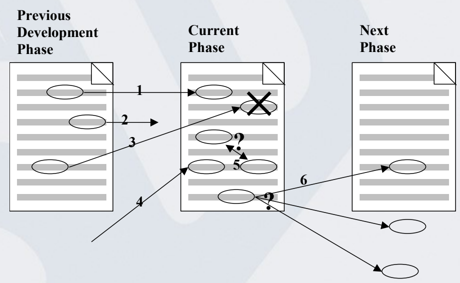
They are introduced due to communication or information transformation issues.

They persist into the developed and deployed software systems

Most of them can be found into those software parts rarely used/executed .

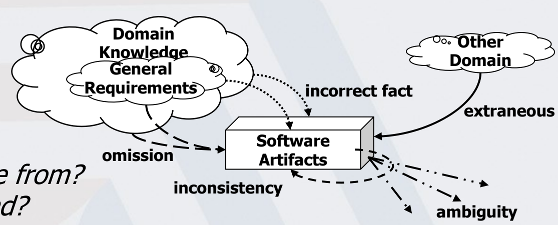
In a generic sense, defects arise when the development work doesn't match software specifications already developed or would cause problems downstream.

1. Information transformed correctly.
2. Information lost during transformation.
3. Information transformed incorrectly.
4. Extraneous information introduced.
5. Multiple inconsistent transformations occurred for same info.
6. Multiple inconsistent transformations possible for same info.



Travassos, G. H., Shull, F. and Carver, J. Working with UML: A Software Design Process Based on Inspections for the Unified Modeling Language, in Advances in Computers, vol. 54, Academic Press, 2001

Software Defects



From where defects come from?
What types of defects we can find?

Defect	***General Description***
Omission	Necessary information about the system has been omitted from the software artifact.
Incorrect Fact	Some information in the software artifact contradicts information in the requirements document or the general domain knowledge.
Inconsistency	Information within one part of the software artifact is inconsistent with other information in the software artifact.
Ambiguity	Information within the software artifact is ambiguous, i.e. any of a number of interpretations may be derived that should not be the prerogative of the developer doing the implementation.
Extraneous Information	Information is provided that is not needed or used.

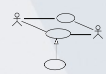
Travassos, G. H., Shull, F. and Carver, J. Working with UML: A Software Design Process Based on Inspections for the Unified Modeling Language, in Advances in Computers, vol. 54, Academic Press, 2001

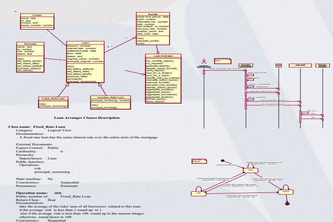
Software Defects

Main cause:

Information mistakenly transformed by developers.

...
3 – The gas station owner can use the system to control inventory. The system will either warn of low inventory or automatically order new parts and gas.
...





class parts inherit from Stock items;
attributes …
services …..
relationships ...

Specification **high and low level design** **coding**

Our Reality...

It is necessary to fi HOW? eliminate software defe oon as possible!

Software Quality Assurance

Verification:

To assure product's consistency, completeness and correctness in each software life cycle stage and between consecutive life cycle stages

"Are we correctly building the product?"

Validation:

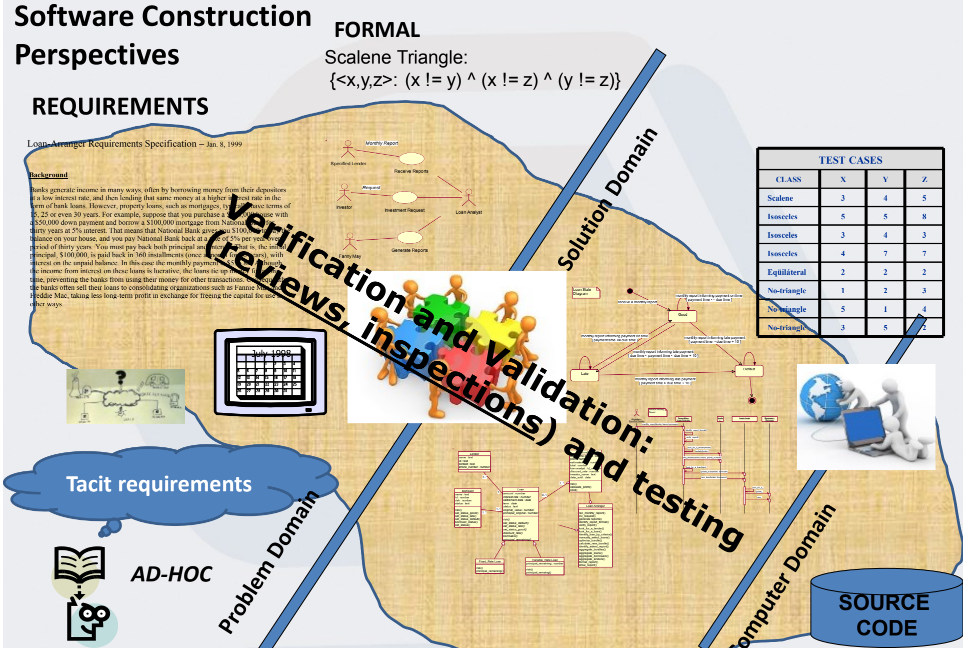
To assure the final product satisfies all software requirements.

"Are we building the correct product?"

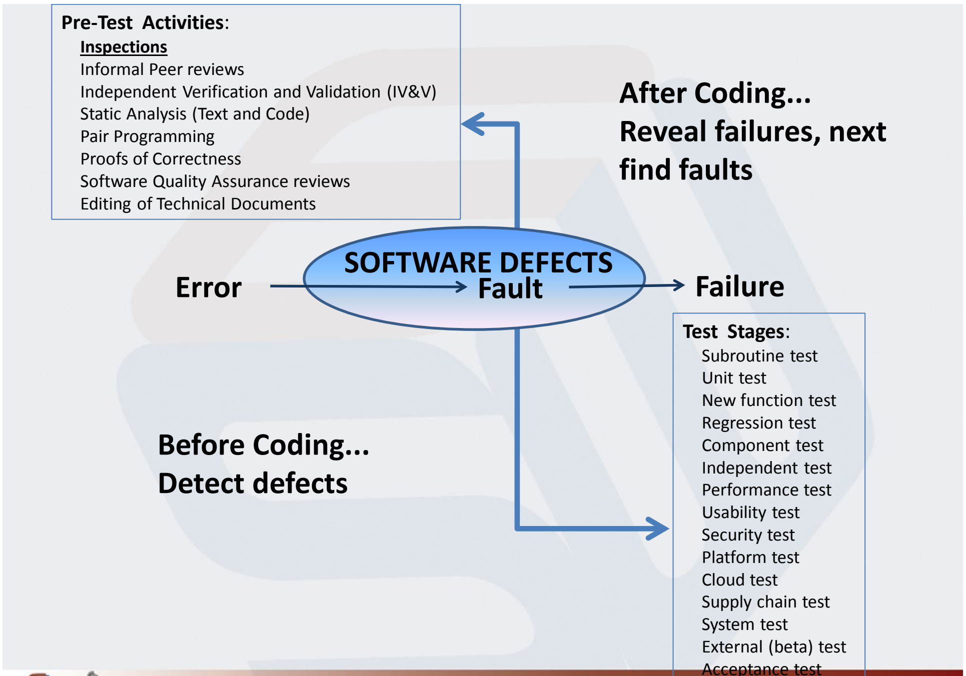
Testing:

To investigate the product behavior by observing the results of its execution.

Software Construction Perspectives



TEST CASES			
CLASS	X	Y	Z
Scalene	3	4	5
Isosceles	5	5	8
Isosceles	3	4	3
Isosceles	4	7	7
Eqüiláteral	2	2	2
No-triangle	1	2	3
No-triangle	5	1	4
No-triangle	3	5	2



Travassos, G.H. (2014). Software Defects: Stay Away from them. Do Inspections!. QUATIC 2014. Keynote. (in press)

Software Quality Assurance

Usually used VV&T activities:

Software review and inspections:

Systematic reading activities performed by the technical staff with the sole objective of finding analysis and design defects produced in the initial phases of development in software artifacts.

Testing:

A multi-step strategy combined with methods for producing representative test cases helping to guarantee effective defect detection.

Patterns and formal procedures: These are patterns and procedures imposed by the client, or rules that direct how the project must be developed.

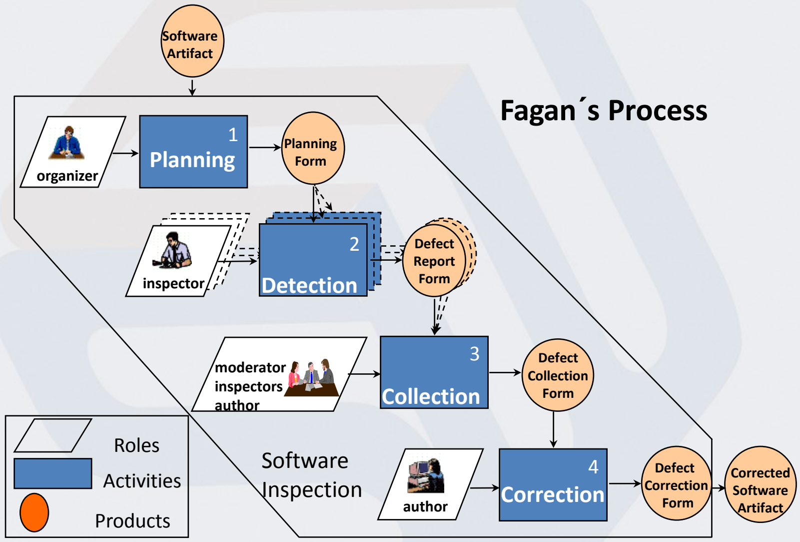
Change control: Contributes to quality by formalizing the order of changes, evaluating the nature of the change and controlling its impact.

Software metrics: Used to trace software quality and to evaluate the impact of various methodologies and procedures.

Registering and keeping of records: Offer information collection and dissemination procedures.

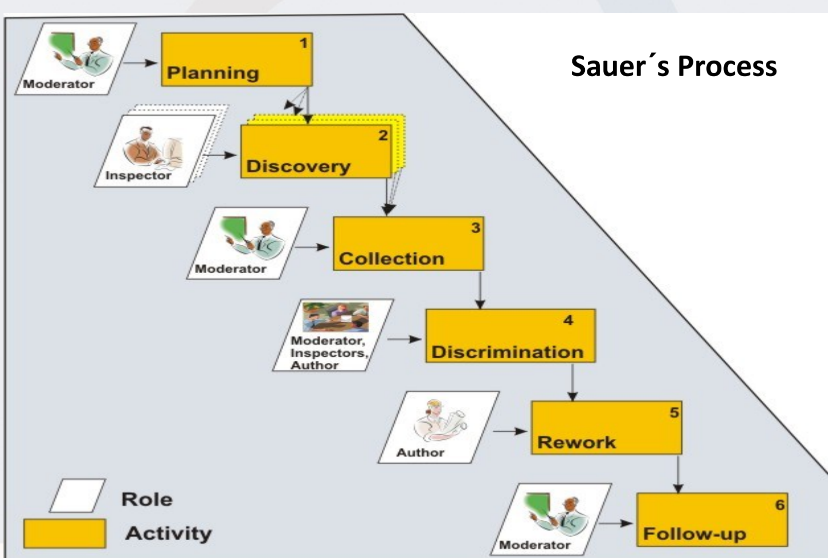
Melo, W.; Shull, F.; Travassos, G.H. (2001). Software Review Guidelines. Systems Engineering and Computer Science Program. COPPE/UFRJ. Technical Report ES-556/01. http://www.cos.ufrj.br/uploadfile/es55601.pdf

Software Inspection Method

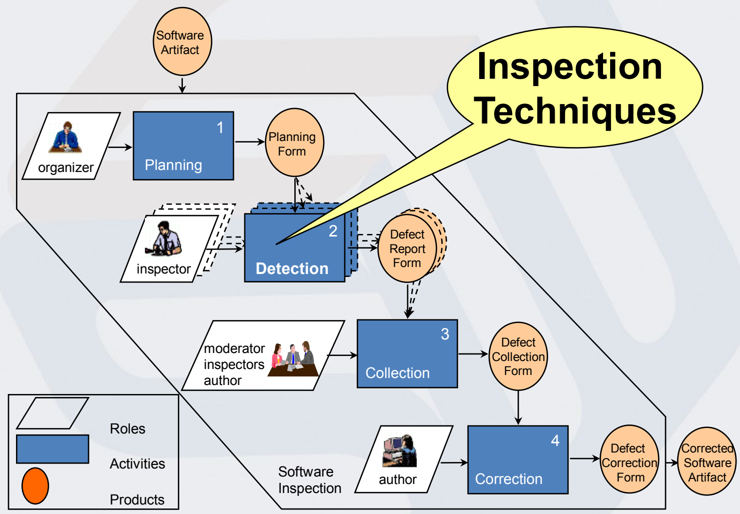


See details in Hernandes, E. M.; Belgamo, A.; Fabbri, S.. (2014). An overview of experimental studies on Software Inspections. Enterprise Information System. Lecture Notes in Businees Information Processing. Vol 190, pp.118-134

Software Inspection Method



Software Inspection Method



Inspection Techniques: *ad-hoc*

Inspector reads the document accordingly its own perspective and knowledge

Individual experience affects the final results:

- Focus on the inspector expertise
- Individual productivity
- Hard to guarantee the inspector read the document in the correct way because each inspector applies its own review approach

There is no document coverage guarantee

Cost/efficiency (#defects/time of inspection) tend to be better when inspectors have high experience (> inspection cost)

Inspection Techniques: checklist

Inspector must follow a list of items representing the software characteristics although following an ad hoc approach (checklists describe what to look for, but not how to look for)

More directed final result:

- Quality characteristics defined *a priori*
- Individual productivity
- Hard to guarantee the inspector reads the document in the correct way even defining the quality characteristics to be reviewed, because each inspector applies its own review approach

Document coverage concerned with the checklist items and inspector approach

Cost/efficiency depends on the checklist and inspectors

Checklist can be tailored or specifically built to capture a specific quality characteristic

Inspection Techniques: checklist

Example: Design Completeness

Inspection Questions	Yes (Pass)	No (Fail)
Package Designs: Does the SDD document all significant package design decisions?		
Unit Designs: Does the SDD document all significant unit design decisions?		
Thoroughly Documented: Are design decisions for the current release documented as completely and as thoroughly as is known at the present time? Note that information relevant to future releases need not be completely documented.		
Current TBDs: Is the acronym "TBD" used to signify that the associated design decisions have not yet been determined and documented?		
No TBDs at Release: Does the final SDD for a release not contain any "TBDs" for that release?		

Software Design Document (SDD) Inspection Checklist – OPEN Process Framework
http://www.opfro.org/index.html?Components/WorkProducts/DesignSet/SoftwareDesignDocument/SoftwareDesignDocumentInspectionChecklist.html~Contents

Inspection Techniques: checklist

Defect Report form
Name: J.J. XPT
Used Checklist: 01
Reviewed Document: Specification Requirements for the USE CASE Tool to support PBR.
Inspection time: 2 hs

Defect No.	Page No.	Req. No.	Defect Type	Description
1	2	RF 8	Omission	Missing a facility to allow the consulting of elements model, such as folders and hierarchical trees.
2			Omission	The requirements do not deal with defects treatments
3	3	RF 11/12	Ambiguity	It is not clear the difference between requirements 11 and 12
4	2	RF 5	Ambiguity	The terms participant and actor are being used to represent the same concept.
5			Omission	It is missing a specification for the user interface and the navigation mechanisms

Inspection Techniques: scenario-based reading

Inspector receives a concrete set of instructions explaining how to read and what to look for in a software product.

- Increase the cost-effectiveness of inspections

More directed final result:

- Quality characteristics and reading approach defined *a priori*
- Technique induces productivity by reducing human influence on inspection results (i.e., ensure a more engineering approach)
- Provide models for writing documents of higher quality
- Easier to guarantee the inspector read the document in the correct way

Document coverage concerned with the reading technique

Cost/efficiency affected by the reading technique

Inspection Techniques: scenario-based reading

More specifically, software reading is the individual analysis of a software artifact (e.g., requirements, design, code, test plans) to achieve the understanding needed for a particular task (e.g., defect detection, reuse, maintenance)

Scenario-based reading is:

document and notation specific

goal driven

tailorable to the project and environment

procedurally defined

focused to provide a particular document coverage

empirically verified to be effective for its use in inspections

Inspection Techniques: scenario-based reading

Different Software Artifacts, Different Reading Techniques

perspective based reading (**PBR**):

for detecting defects in requirements documents

traceability based (horizontal/vertical) reading (**OORTS**):

for detecting defects in object oriented design in UML

usability based (heuristics) reading (**WDP**):

for detecting anomalies in user interface web screens

defect based reading (**DBR**):

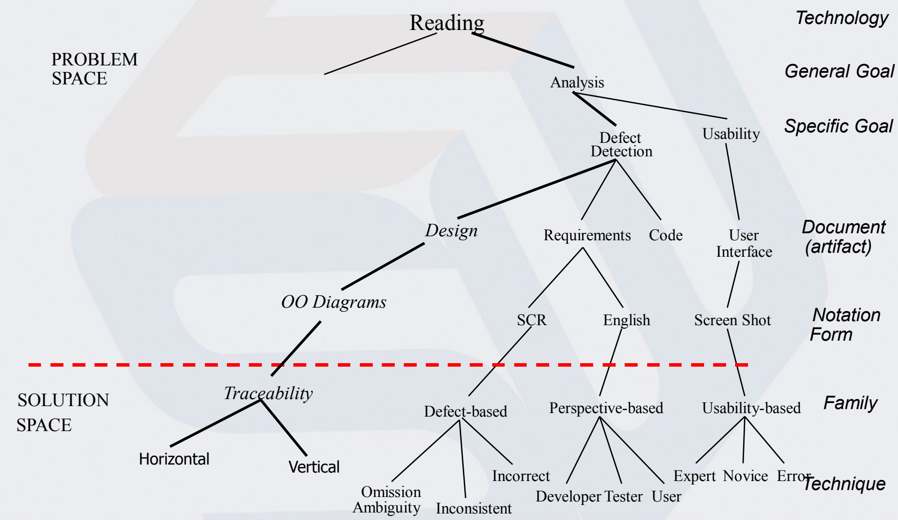
for detecting defects in requirements documents in SCR

scope based reading:

for constructing designs from OO frameworks

Reading techniques define an *approach* to be *tailored*.
There are different set of reading techniques.

Inspection Techniques: scenario-based reading

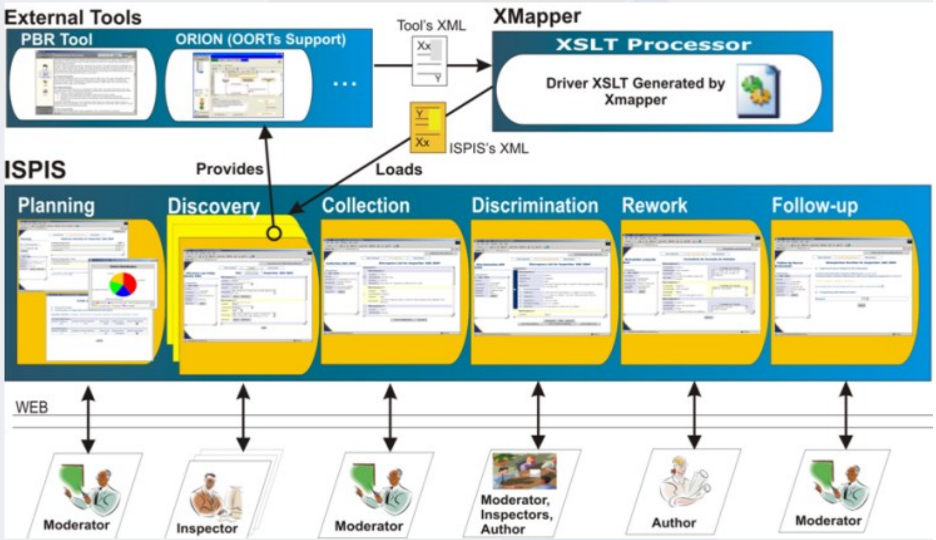


Software Inspection Techniques: summary

Technique / Features	*Ad-hoc*	Checklist based	Scenario-based reading
Notation	any	any	Language of "doing"
Systematic	no	partially	yes
Focused	no	no	yes
Controlled Improvement	does not allow	partially	yes
Adaptable	no	yes	yes
Training	no	partially	yes
Tailoring	no need	needed whether capturing specific quality characteristics	needed due to the used model
Introduction effort	low	medium	high
Document Coverage	no guarantee	depends on checklist and the inspector approach, but still hard to guarantee	Controlled by the technique
Cost-efficiency	depends on inspectors' experience	depends on inspectors' experience and checklist	depends on the technique, usually high

Travassos, G.H. (2014). Software Defects: Stay Away from them. Do Inspections!. QUATIC 2014. Keynote. (in press)

Software Inspection Tool



Kalinowski, M. ; Travassos, G. H. (2004). A Computational Framework for Supporting Software Inspections. In: IEEE 19th International Conference on Automated Software Engineering - ASE'04, IEEE Computer Press, v. 1. p. 46-55.

Evidence on Software Inspections
(academia)

Inspections significantly increase productivity, quality, and project stability.
Fagan´s law

Effectiveness of Inspections is fairly independent of its organizational form.
Porter-Votta's law

Perspective-based inspections are (highly) effective and efficient.
Basili´s law

A combination of different V&V methods outperforms any single method alone.
Hetzel-Myers law

Endres, A; Rombach, D. (2003). A Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories. Fraunhofer IESE Series on Software Engineering. Pearson/Addison Wesley.ISBN 0321154207

Evidence on Software Inspections
(academia)

- Quality entails productivity.
  - Mills-Jones hypothesis
- Error prevention is better than error removal.
  - May's hypothesis
- Proving of programs solves the problems of correctness, documentation, and compatibility.
  - Hoare's hypothesis
- Approximately 80 percent of defects come from 20 percent of modules.
  - Pareto–Zipf-type laws

Endres, A; Rombach, D. (2003). A Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories. Fraunhofer IESE Series on Software Engineering. Pearson/Addison Wesley.ISBN 0321154207

Evidence on Software Inspections
(industry)

Company	Software Category	Inspected Artifact	Results
AT&T	Telecom	Requirements, design, code and testing	Inspection has increased productivity and quality by 14%, being 20x more efficient than testing.
HP	Varied	Design, code, testing, documentation	An audit revealed an ineffective inspection process. Problems under discussion.
		Code	2 defects detected per hour. It is unlikely that 80% of defects could be caught by testing.
BRN	Telecom	Code	1 defect detected per hour. The process was 20x more efficient than testing.
Bull HN Information Systems	Operating system	Requirements, design, code, testing, documentation.	4 people's teams were twice as efficient as the one composed of 3.

Travassos, G.H. (2014). Software Defects: Stay Away from them. Do Inspections!. QUATIC 2014. Keynote. (in press)

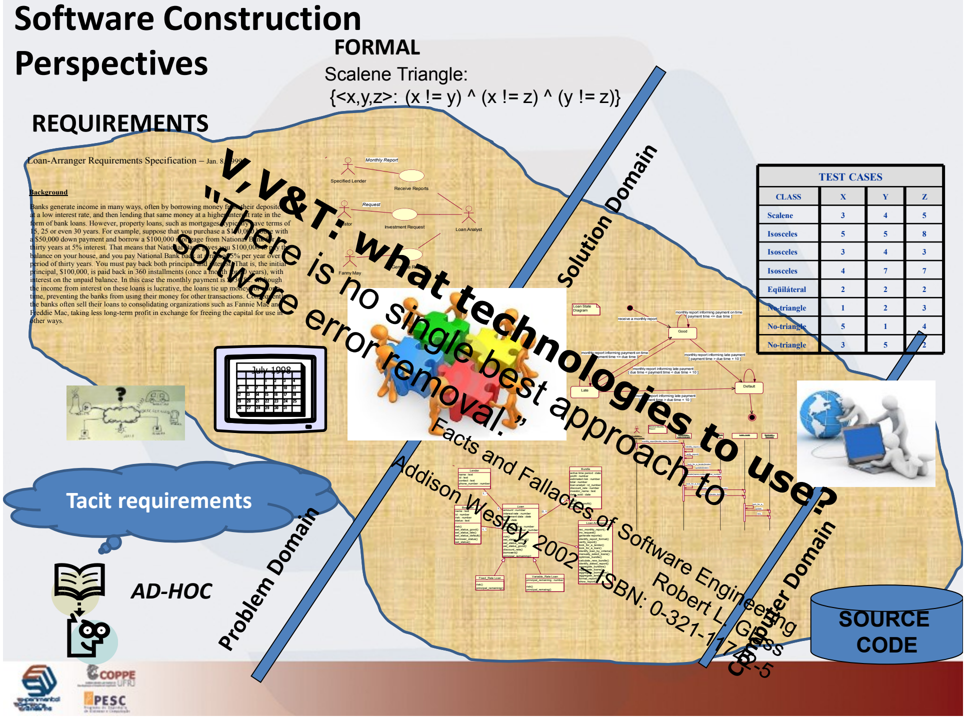
Evidence on Software Inspections (industry)

Company	Software Category	Inspected Artifact	Results
IBM	Operating system	Design and code	23% increasing in code productivity and 38% reduction of defects found in test stage.
ICL	Operating system	Design	40% to 50% increasing in defect detection. 1.2 hours per defect in inspection compared to 8.4 hours with testing.
JPL	Space system	Requirements, design, code, testing	0.5 hours to find defects versus 5 hours for other techniques.
MEL	Varied	Design, code	ROI calculated at 8:1. In 75 inspections the result was 7000 hours saved.
Shell Research	Geophysical software	Requirements	1 defect found every 3 minutes. Return on investment calculated at 30:1.

COPPE UFRJ PESC

Travassos, G.H. (2014). Software Defects: Stay Away from them. Do Inspections!. QUATIC 2014. Keynote. (in press)

Software Construction Perspectives



COPPE UFRJ PESC

Verification, Validation and Testing

Pre-Test Activities Efficiency

Artifacts / Activity	Architecture	Requirements	Design	Source Code	Document
Inspections					
Requirement	5%	**87%**	10%	5%	8.5%
Architecture	**85%**	10%	10%	2.5%	12%
Design	10%	14%	**87%**	7%	16%
Code	12.5%	15%	20%	**85%**	10%
Static Analysis	2%	2%	7%	**87%**	3%
IV&V	10%	12%	23%	7%	**18%**
SQA Review	10%	**17%**	**17%**	12%	12.5%

1.Adapted from Capers Jones. (2014). The Ranges and Limits of Software Quality. Available at http://Namcookanalytics.com.

Verification, Validation and Testing

Test Stages Efficiency

Artifacts / Testing Stages	Architecture	Requirements	Design	Source Code	Document
Unit	2.5%	4%	7%	**35%**	10%
Function	7.5%	5%	22%	**37.5%**	10%
Regression	2%	2%	5%	**33%**	7.5%
Integration	6%	20%	22%	**33%**	15%
Performance	14%	2%	**20%**	18%	2.5%
Security	12%	15%	**23%**	8%	2.5%
Usability	12%	17%	15%	5%	**48%**
System	16%	12%	18%	12%	**34%**
Cloud	10%	5%	13%	10%	**20%**
Independent	12%	10%	11%	10%	**23%**
Field (Beta)	14%	12%	14%	12%	**34%**
Acceptance	13%	14%	15%	12%	**24%**

1.Adapted from Capers Jones. (2014). The Ranges and Limits of Software Quality. Available at http://Namcookanalytics.com.

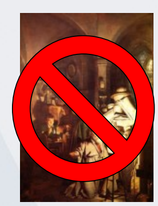
Final Remarks

- Software Technology decisions shall be based on evidence.
- Investigations in software engineering share some of the same issues as social science (inspired on...):
  - difficult to collect data
  - non-repeatable
  - difficult to control
- The more we care with defect removal
  - the more confidence we can have in the quality of our products
  - the better can be our projects
  - the more effective will be our actions

Conclusion

There is no silver bullet!!

There is no philosopher's stone!!



Your mission: TO DETECT AND REMOVE DEFECTS!

Learn with them!!!!

Promote inspections as much you can and permit <u>moderated</u> empiricism to support your research, development and decision making:

it can help to reduce software systems fails and contribute to the advance of the field.

58

Software defects:

Stay Away from them.

Do Inspections!

Obrigado por sua atenção.

Guilherme Horta Travassos
Universidade Federal do Rio de Janeiro
COPPE/PESC
CNPq Researcher, **ISERN** Member
ght@cos.ufrj.br
www.cos.ufrj.br/~ght

QUATIC 2014