Software Improvement Group

Software Analysis and Testing

Métodos Formais em Engenharia de *Software*

December 2010
Joost Visser

info@sig.nl
www.sig.nl

Me

CV

- Technical University of Delft, Computer Science, MSc 1997
- University of Leiden, Philosophy, MA 1997
- CWI (Center for Mathematics and Informatics), PhD 2003
- Software Improvement Group, developer, consultant, etc, 2002-2003
- Universidade do Minho, Post-doc, 2004-2007
- Software Improvement Group, Head of Research, 2007-…

Research

- Grammars, traversal, transformation, generation
- Functional programming, rewriting strategies
- Software quality, metrics, reverse engineering

Software Improvement Group

Company

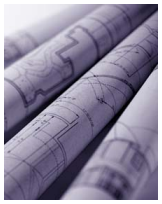
- Spin-off from CWI in 2000, self-owned, independent
- Management consultancy grounded in source code analysis
- Innovative, strong academic background, award-winning, profitable

Services

- Software Risk Assessments (snapshot) and Software Monitoring (continuous)
- Toolset enables to analyze source code in an automated manner
- Experienced staff transforms analysis data into recommendations
- We analyze over 50 systems annually
- Focus on technical quality, primarily maintainability / evolvability

Services

Software Improvement Group



DocGen

- Automated generation of technical documentation
- Reduce learning time, assist impact analysis, support migration, …

Software Risk Assessment

- In-depth investigation of software quality and risks
- Answers specific research questions

Software Monitoring

- Continuous measurement, feedback, and decision support
- Guard quality from start to finish

Software Product Certification

- Five levels of technical quality
- Evaluation by SIG, certification by TÜV Informationstechnik

Who is using our services?

Software Improvement Group

Financials & insurance companies

Public

Retail/Logistics

Technology

Utilities/Telco

Structure of the lecture

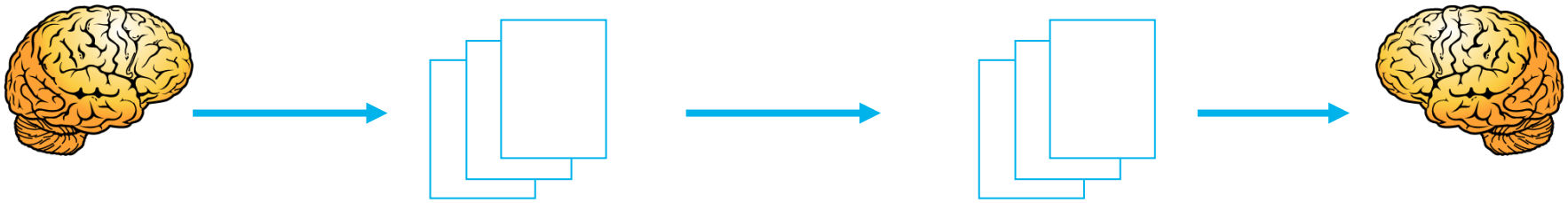
- Introduction SIG

- General overview of software analysis and testing
- Testing
- Patterns
- Quality & metrics
- Reverse engineering

Software Improvement Group

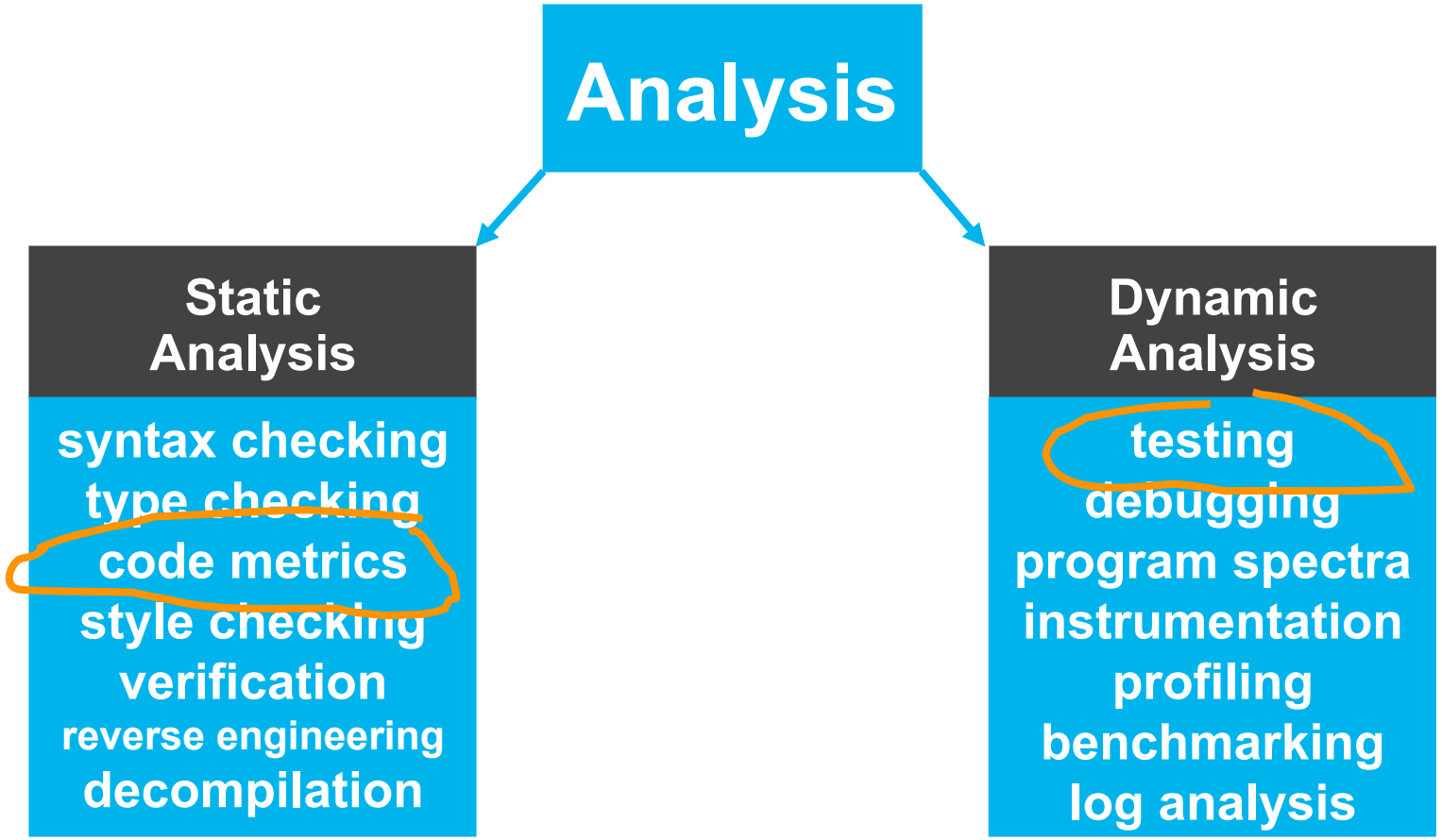
Create → **Change** → **Analyze**

Create	Change	Analyze
requirements analysis design, code, compile configure, install	refactor, fix, patch maintain, renovate evolve, update, improve	understand, assess evaluate, test measure, audit



Software Analysis (and Testing)

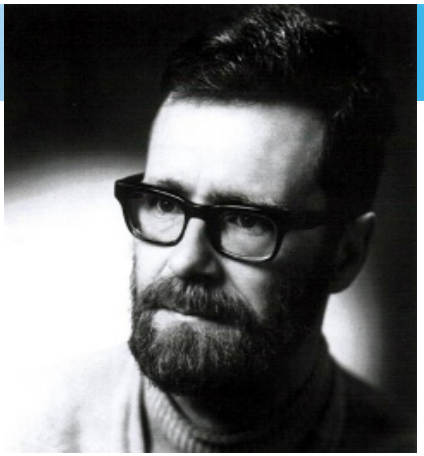
Software Improvement Group



Is testing un-cool?

Software Improvement Group

Edsger Wybe Dijkstra (1930 - 2002)



- "Program testing can be used to show the presence of bugs, but never to show their absence!"
  Notes On Structured Programming, 1970

- "Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence."
  The Humble Programmer, ACM Turing Award Lecture, 1972

Does not mean: "Don't test!!"

Is testing un-cool?

Software Improvement Group

Industry

- Testers earn less then developers
- Testing is “mechanical”, developing is “creative”
- Testing is done with what remains of the budget in what remains of the time

Academia

- Testing is not part of the curriculum, or very minor part
- Verification is superior to testing
- Verification is more challenging than testing

Software Analysis. How much?

Software Improvement Group

50 - 75%

In a typical commercial development organization, the cost of providing [the assurance that the program will perform satisfactorily in terms of its functional and nonfunctional specifications within the expected deployment environments] via appropriate debugging, testing, and verification activities can easily range from 50 to 75 percent of the total development cost. (Hailpern and Santhanam, 2002)

Software Analysis. Enough?

Software Improvement Group

Planning Report 02-3

The Economic Impacts of Inadequate Infrastructure for Software Testing

$\$60 \times 10^9$

Table ES-4. Costs of Inadequate Software Testing Infrastructure on the National Economy

	The Cost of Inadequate Software Testing Infrastructure (billions)	Potential Cost Reduction from Feasible Infrastructure Improvements (billions)
Software developers	$21.2	$10.6
Software users	$38.3	$11.7
Total	$59.5	$22.2

of total impacts, and software users accounted for the about 60 percent.

Software Analysis. More?

Software Improvement Group

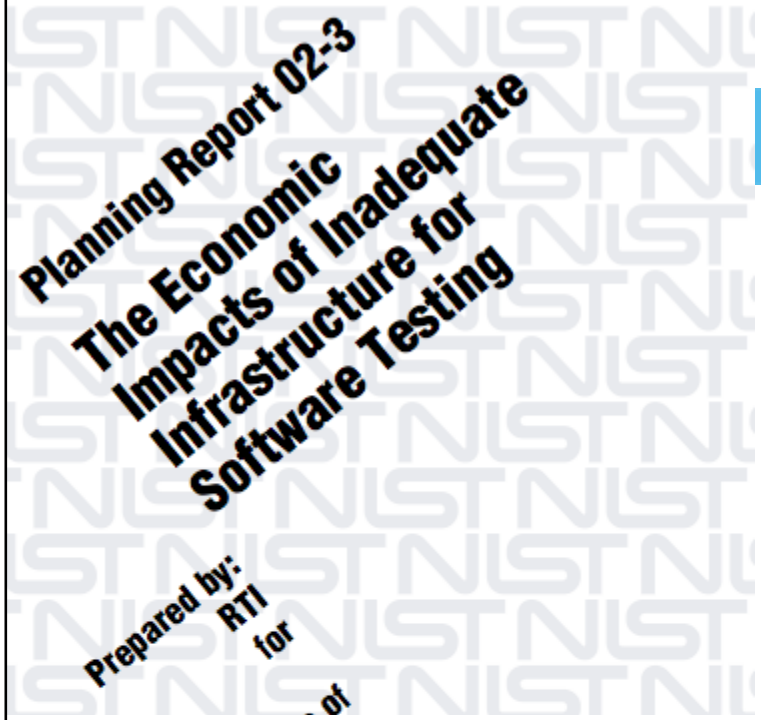
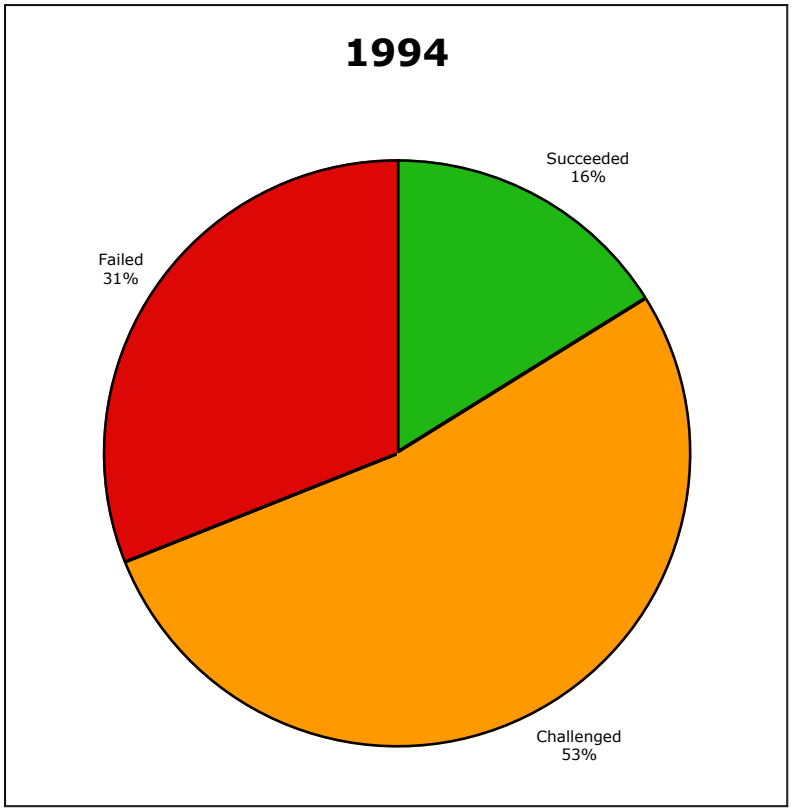


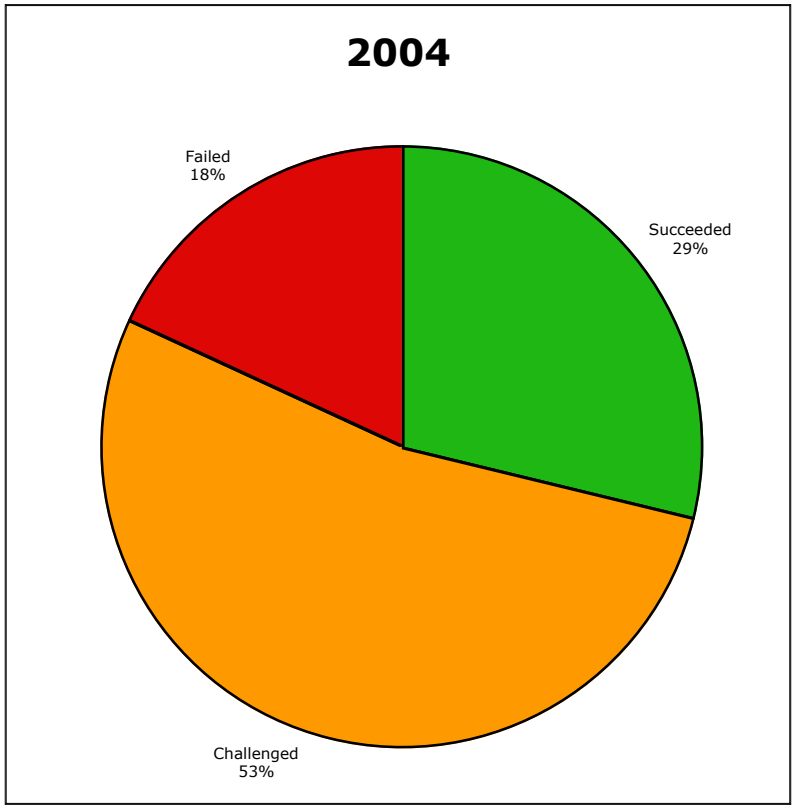
Table 1-4. Recent Aerospace Losses due to Software Failures

	Airbus A320 (1993)	Ariane 5 Galileo Poseidon Flight 965 (1996)	Lewis Pathfinder USAF Step (1997)	Zenit 2 Delta 3 Near (1998)	DS-1 Orion 3 Galileo Titan 4B (1999)
Aggregate cost		$640 million	$116.8 million	$255 million	$1.6 billion
Loss of life	3	160			
Loss of data		Yes	Yes	Yes	Yes

Software Analysis
Room for improvement?

Software Improvement Group





Standish Group, *"The CHAOS Report"*

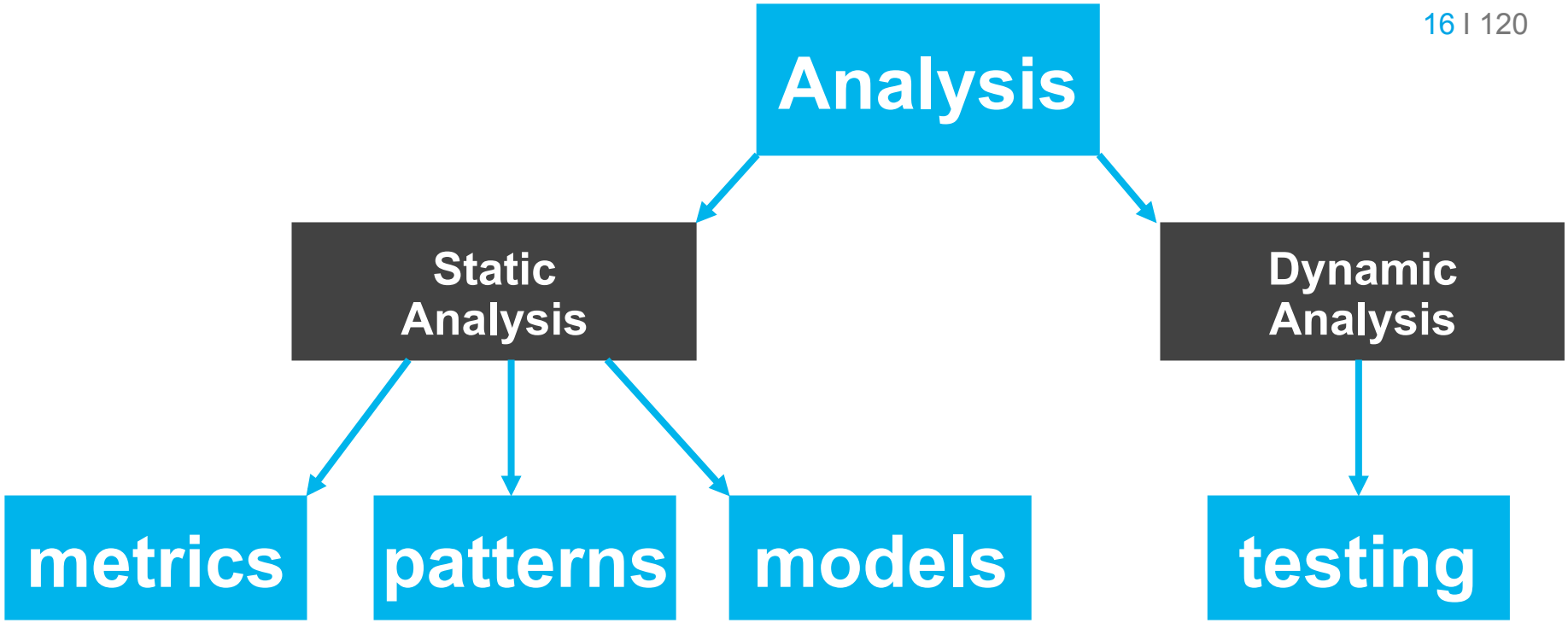
So

- Testing ⊂ Dynamic analysis ⊂ Analysis ⊂ S.E.
- Analysis is a major and essential part of software engineering
- Inadequate analysis costs billions

- More effective and more efficient methods are needed
- Interest will keep growing in both industry and research

Structure of the lectures

Software Improvement Group



TESTING

Testing

Kinds

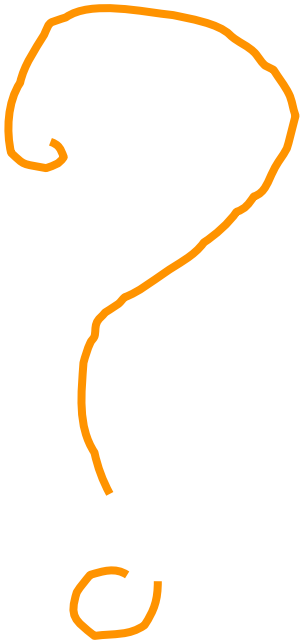
- Conformance
- Interoperability
- Performance
- Functional
- White-box
- Black-box
- Acceptance
- Integration
- Unit
- Component
- System
- Smoke
- Stress

Ways

- Manual
- Automated
- Randomized
- Independent
- User
- Developer

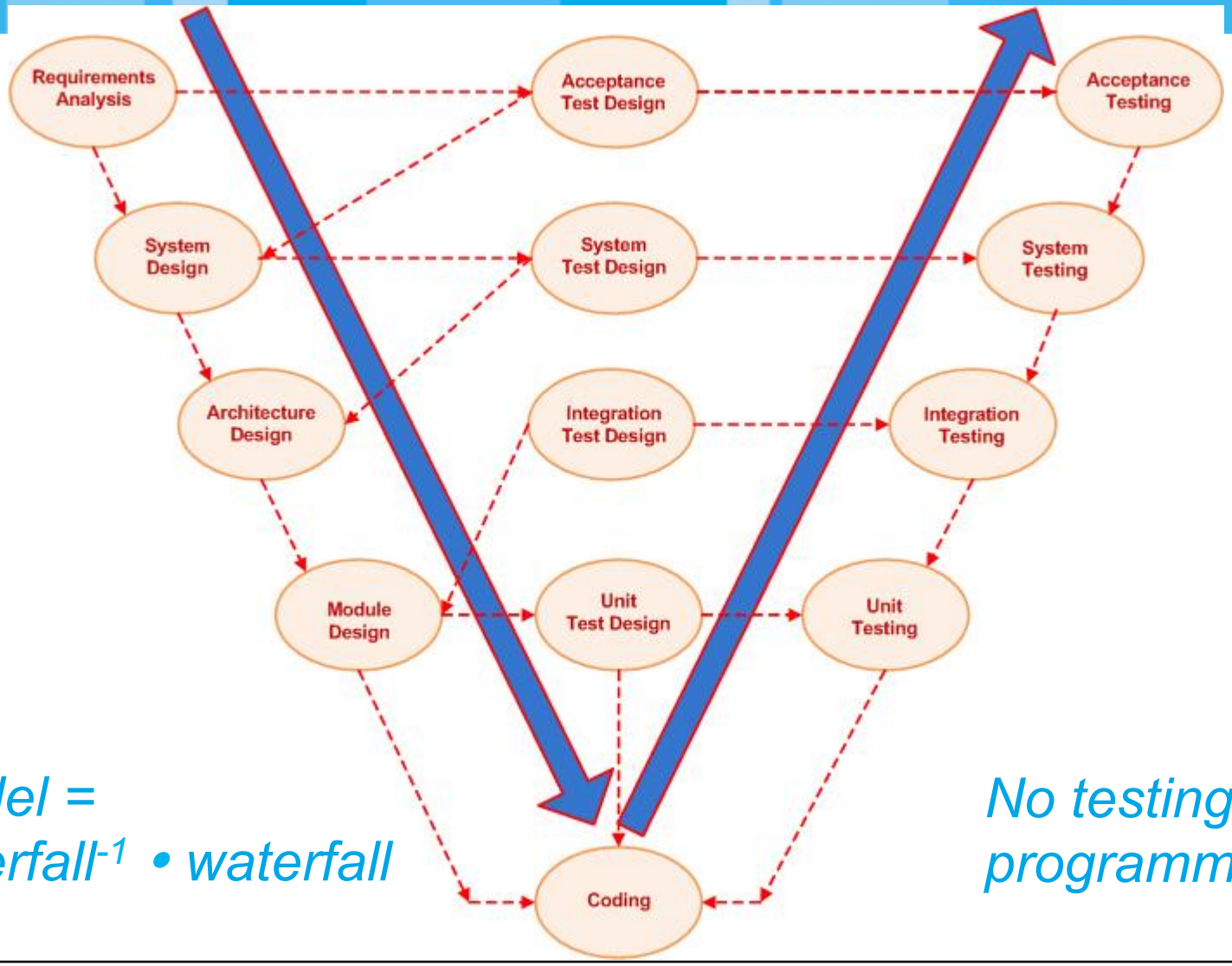
With

- Plans
- Harness
- Data
- Method
- Frameworks



Testing
V-model

Software Improvement Group



V-model = waterfall^{-1} • waterfall

No testing while programming!

Testing
Eliminate waste

Waste

- Coding and debugging go hand-in-hand
- Coding effort materializes in the delivered program
- Debugging effort? Evaporates!

Automated tests

- Small programs that capture debugging effort.
- Invested effort is consolidated …
- … and can be re-used without effort ad-infinitum

Unit testing

What is unit testing?

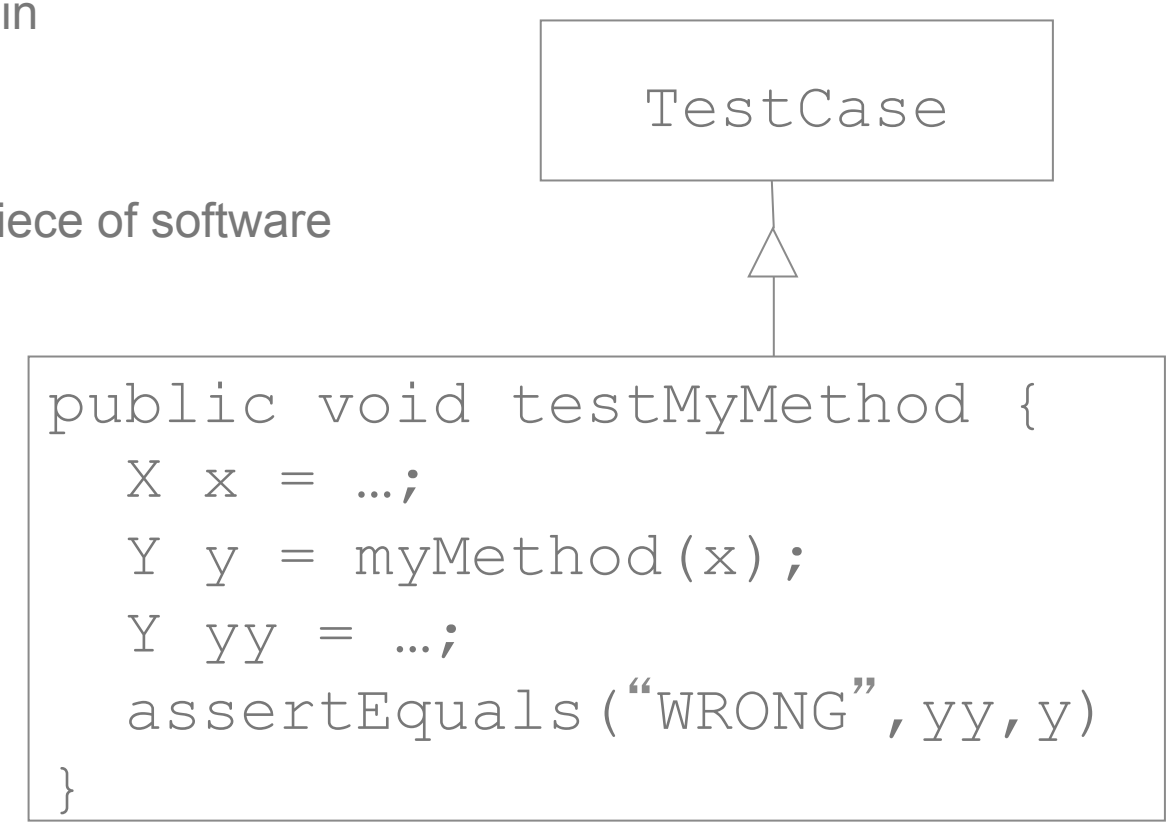
Software Improvement Group

A unit test is …

- fully automated and repeatable
- easy to write and maintain
- non-intrusive
- documenting
- applies to the simplest piece of software

Tool support

- JUnit and friends



Testing goals

Unit testing has the following goals:

- Improve quality
  - Test as specification
  - Test as bug repellent
  - Test as defect localization
- Help to understand
  - Test as documentation
- Reduce risk
  - Test as a safety net
  - Remove fear of change

Observing unit-testing maturity in the wild (characterization of the population)

Organization

- public, financial, logistics
- under contract, in house, product software
- with test departments, without test departments

Architecture & Process

- under architecture, using software factories
- model driven, handwritten
- open source frameworks, other frameworks
- using use-cases/requirements
- with blackbox tools, t-map

Technology

- information systems, embedded
- webbased, desktop apps
- java, c#, 4GL's, legacy
- latest trend: in-code asserts (java.spring)

Stage 1
No unit testing

Observations:

- Very few organizations use unit testing
- Also brand new OO systems without any unit tests
- Small software shops and internal IT departments
- In legacy environments: programmers describe in words what tests they have done.

Symptoms:

- Code is instable and error-prone
- Lots of effort in post-development testing phases

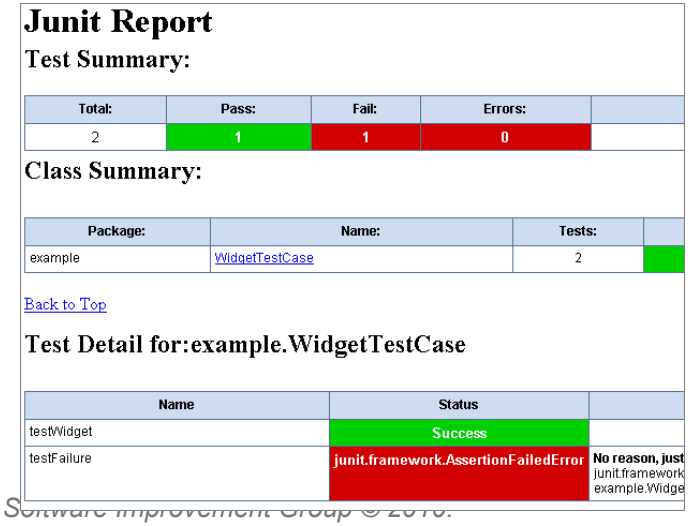
Stage 1
No unit testing

Excuses:

- “It is just additional code to maintain”
- “The code is changing too much”
- “We have a testing department”
- “Testing can never prove the absence of errors”
- “Testing is too expensive, the customer does not want to pay for it”
- “We have black-box testing”

Action

- Provide standardized framework to lower threshold
- Pay for unit tests as deliverable, not as effort



Junit Report

Test Summary:

Total:	Pass:	Fail:	Errors:	
2	1	1	0	

Class Summary:

Package:	Name:	Tests:	
example	WidgetTestCase	2	

Back to Top

Test Detail for:example.WidgetTestCase

Name	Status	
testWidget	Success	
testFailure	junit.framework.AssertionFailedError	**No reason, just** junit.framework example.Widge

Stage 2
Unit test but no coverage measurement

Observations

- Contract requires unit testing, not enforced
- Revealed during conflicts
- Unit testing receives low priority
- Developers relapse into debugging practices without unit testing
- Good initial intentions, bad execution
- Large service providers

Symptoms:

- Some unit tests available
- Excluded from daily build
- No indication when unit testing is sufficient
- Producing unit test is an option, not a requirement

Stage 2
Unit test but no coverage measurement

Software Improvement Group

Excuses:

- “There is no time, we are under pressure”
- “We are constantly stopped to fix bugs”

Actions

- Start measuring coverage
- Include coverage measurement into nightly build
- Include coverage result reports into process

Stage 3
Coverage, not approaching 100%

Observations

- Coverage is measured but gets stuck at 20%-50%
- Ambitious teams, lacking experience
- Code is not structured to be easily unit-testable

Symptoms:

- Complex code in GUI layer
- Libraries in daily build, custom code not in daily build

Stage 3
Coverage, not approaching 100%

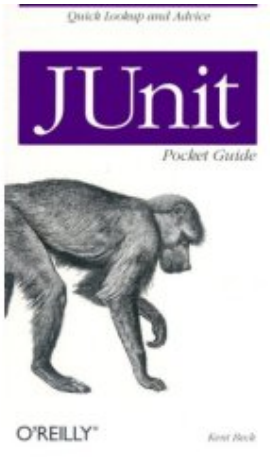
Software Improvement Group

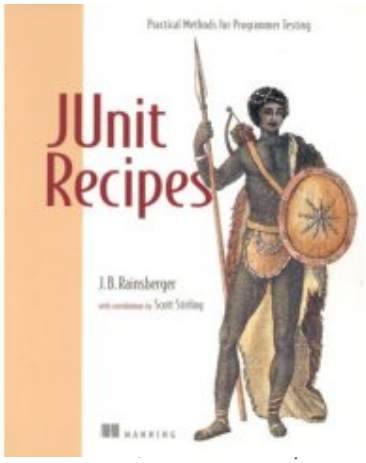
Excuses

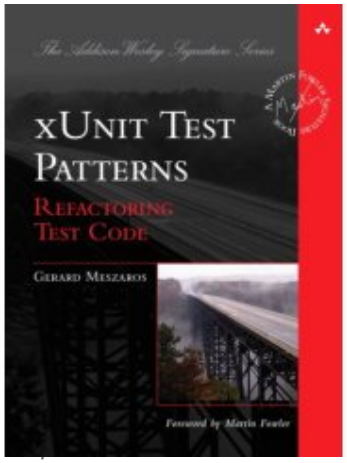
- “we test our libraries thoroughly, that affects more customers”

Actions:

- Refactor code to make it more easily testable
- Teach advance unit testing patterns
- Invest in set-up and mock-up







Stage 4
Approaching 100%, but no test quality

Observations

- Formal compliance with contract
- Gaming the metrics
- Off-shored, certified, bureaucratic software factories

Symptoms:

- Empty tests
- Tests without asserts.
- Tests on high-level methods, rather than basic units

- Need unit tests to test unit tests

Stage 4
Approaching 100%, but no test quality

Anecdotes:

- Tell me how you measure me, and I tell you how I behave
- We have generated our unit tests (at first this seems a stupid idea)

Action:

- Measure test quality
- Number of asserts per unit test
- Number of statements tested per unit test
- Ratio of number of execution paths versus number of tests

Stage 5
Measuring test quality

Enlightenment:

- Only one organization: a Swiss company
- Measure:
  - Production code incorporated in tests
  - number of assert and fail statements
  - low complexity (not too many ifs)
- The process
  - part of daily build
  - “stop the line process”, fix bugs first by adding more tests
  - happy path and exceptions
  - code first, test first, either way

Testing
Intermediate conclusion

Software Improvement Group

Enormous potential for improvement:

- Do unit testing
- Measure coverage
- Measure test quality

- May not help Ariane 5
- Does increase success ratio for “normal” projects

Randomized Testing (quickcheck)

Randomized testing:

- QuickCheck: initially developed for Haskell
- Parameterize tests in the test data
- Property = parameterized test
- Generate test data randomly
- Test each property in 100 different ways each time

Test generation

Model-driven testing

Fault-injection

```
-- | Range of inverse is domain.
prop_RngInvDom r
  = rng (inv r) == dom r
    where
      types = r::Rel Int Integer
```

Is testing un-cool?

Software Improvement Group

Edsger Wybe Dijkstra (1930 - 2002)

- "Program testing can be used to show the presence of bugs, but never to show their absence!"

Martin Fowler

- "Don't let the fear that testing can't catch all bugs stop you from writing the tests that will catch most bugs."

Simple test metrics

Line coverage

- Nr of test lines / nr of tested lines

Decision coverage

- Nr of test methods / Sum of McCabe complexity index

Test granularity

- Nr of test lines / nr of tests

Test efficiency

- Decision coverage / line coverage

Testing
Exercises

Write unit tests

- Using JUnit
- E.g. for one of your own projects

Measure coverage

- E.g. using Emma plug-in for Eclipse

Randomize one of your unit tests

- Turn test into property with *extract method* refactoring
- Write generator for test data
- Instantiate property 100 times with random test data
- Solution to j.visser@sig.eu