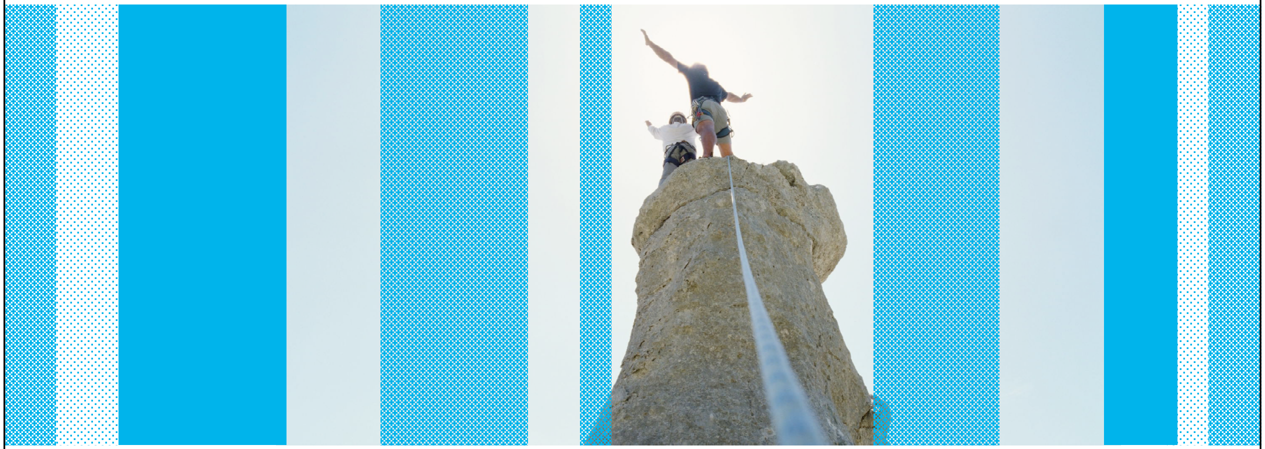


Software Analysis and Testing

Métodos Formais em Engenharia de *Software*

November 2007
Joost Visser

Arent Janszoon Ernststraat 595-H
NL-1082 LD Amsterdam
info@sig.nl
www.sig.nl

Software Improvement Group

Company

- Spin-off from CWI in 2000, self-owned, independent
- Management consultancy grounded in source code analysis
- Winner of the Innovator Award 2007

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

Who is using our services?

Software Improvement Group

Financials / Insurance companies

ABN·AMRO, PGGM, ING, Zorg en Zekerheid, Rabobank, InterBank, achmea, ZwitserLeven, Friesland Bank, delta lloyd, LeasePlan, Allianz

Government

RVR Raden Voor Rechtsbijstand, VROM, RDW, POLITIE, cwi, Ministerie van Binnenlandse Zaken en Koninkrijksrelaties, SVB, kadaster

Logistical

KLM, DHL, TNT, EUROMAX TERMINAL, norfolkline DOVER - DUNKERQUE FERRIES, ProRail

IT

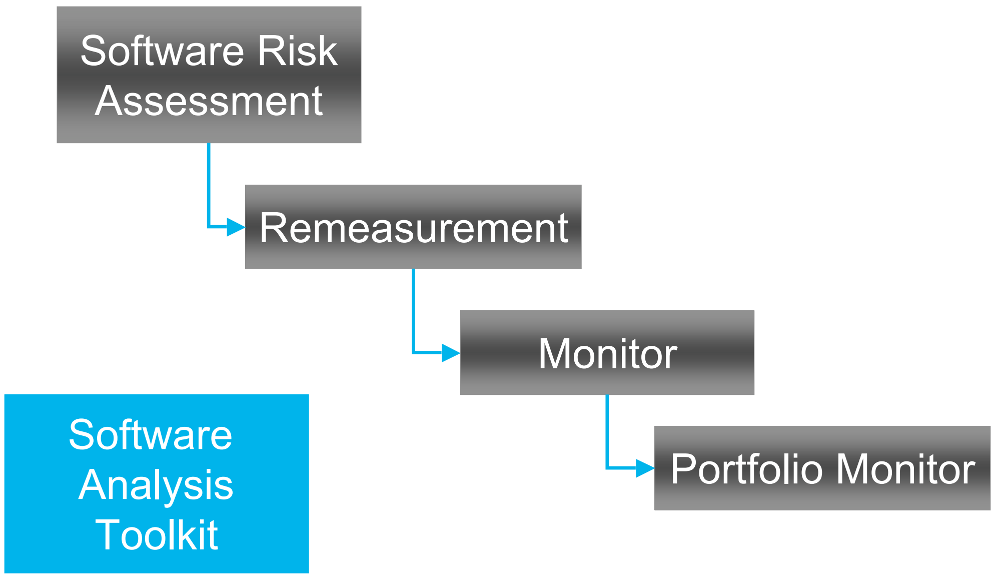
Getronics PinkRoccade, CENTRIC, Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING, Exact software, IBM, CHESS, logicaCMG

Other

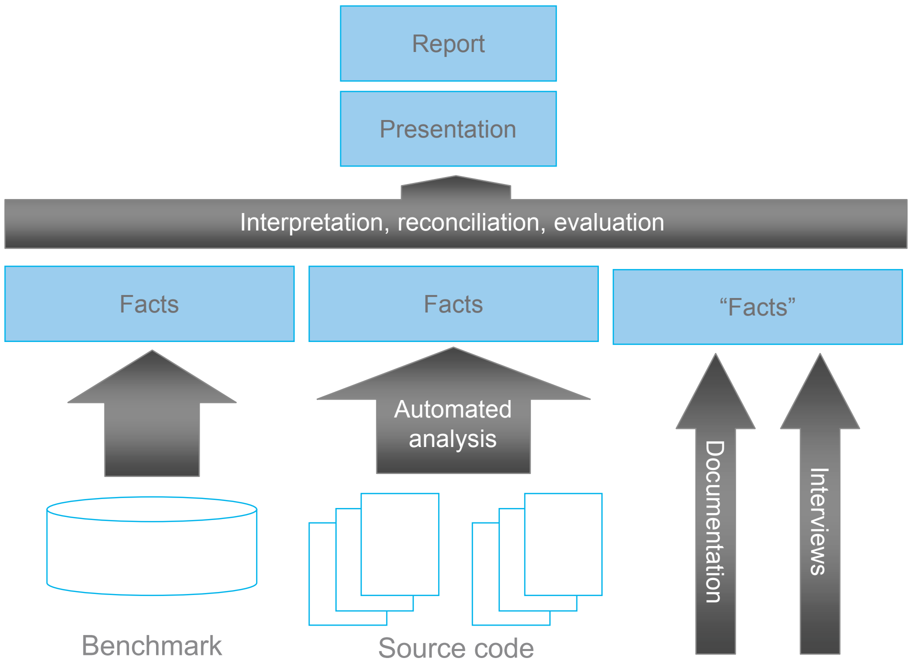
ENECO energie, PRICEWATERHOUSECOOPERS, CARe SCHADESERVICE, KPMG, oxxio, Gasunie

Our services

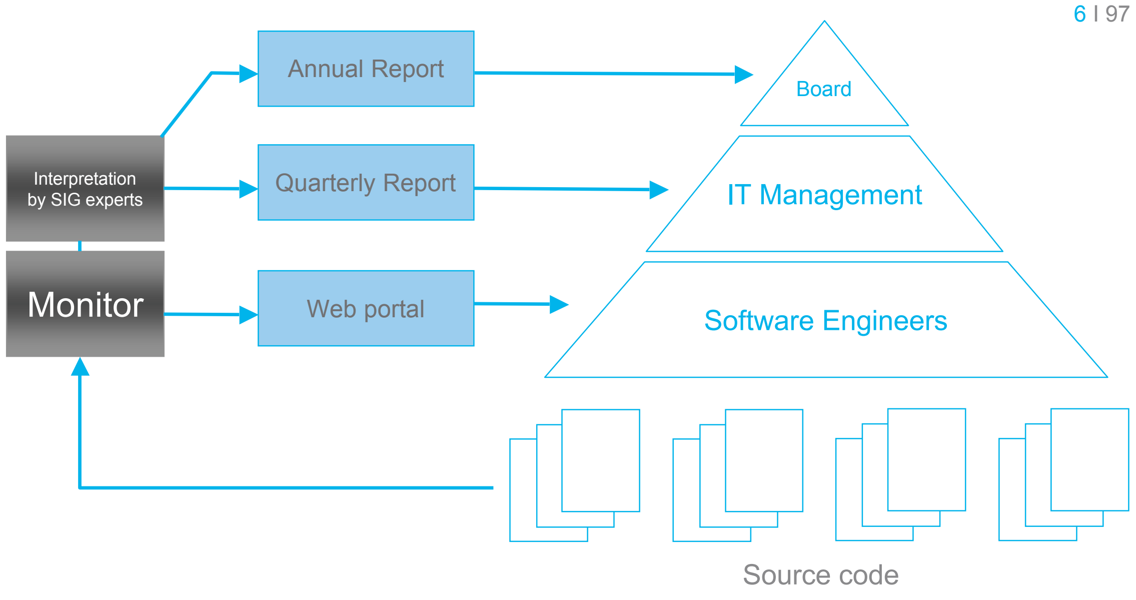
Software Improvement Group



Software Risk Assessment



Software Quality Monitor



Structure of the lectures

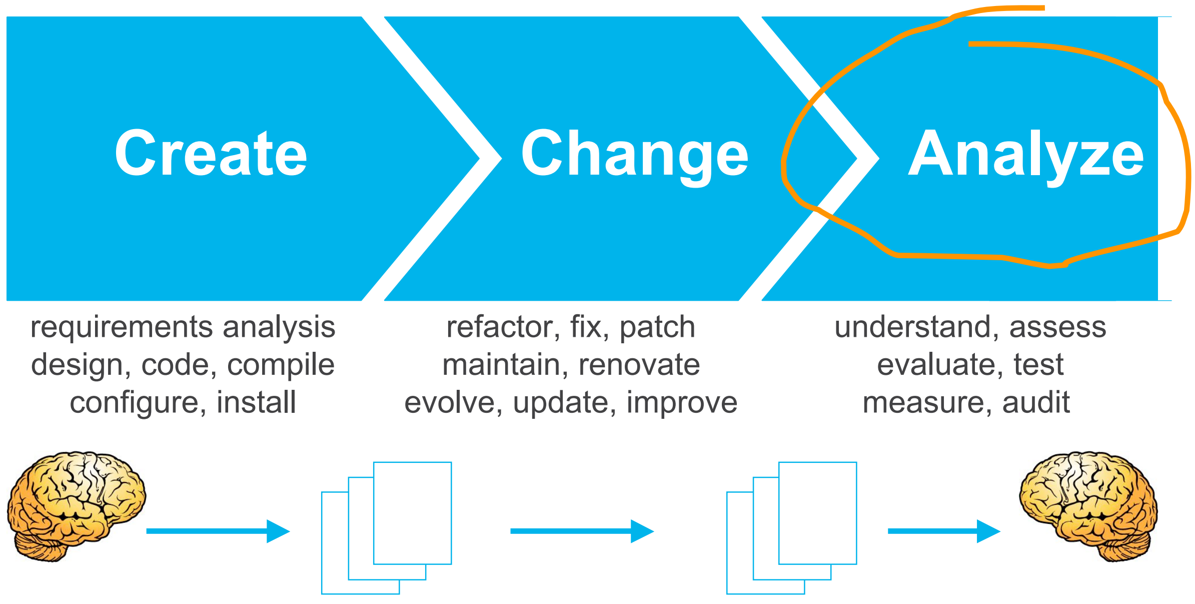
Today

- Introduction SIG
- General overview of software analysis and testing
- Testing
- Patterns

Next week

- Quality & metrics
- Reverse engineering
- …

Software Engineering



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)

Analysis

Static Analysis

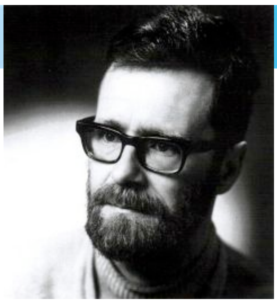
- syntax checking
- type checking
- code metrics
- style checking
- verification
- reverse engineering
- decompilation

Dynamic Analysis

- testing
- debugging
- program spectra
- instrumentation
- profiling
- benchmarking
- log analysis

Is testing un-cool?

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?

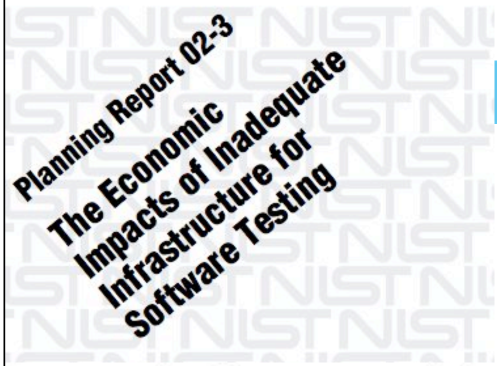
Industry

- 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

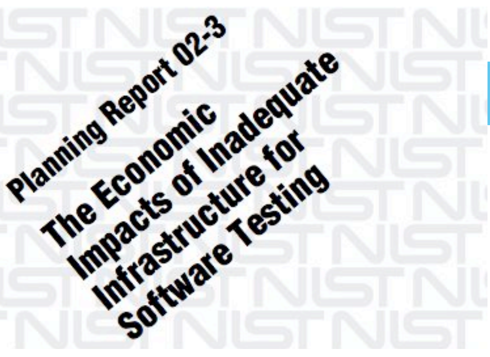
Software Analysis. How much?


Planning Report 02-3
The Economic Impacts of Inadequate Infrastructure for Software Testing

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?



Planning Report 02-3

The Economic Impacts of Inadequate Infrastructure for Software Testing

$60 × 10^9$

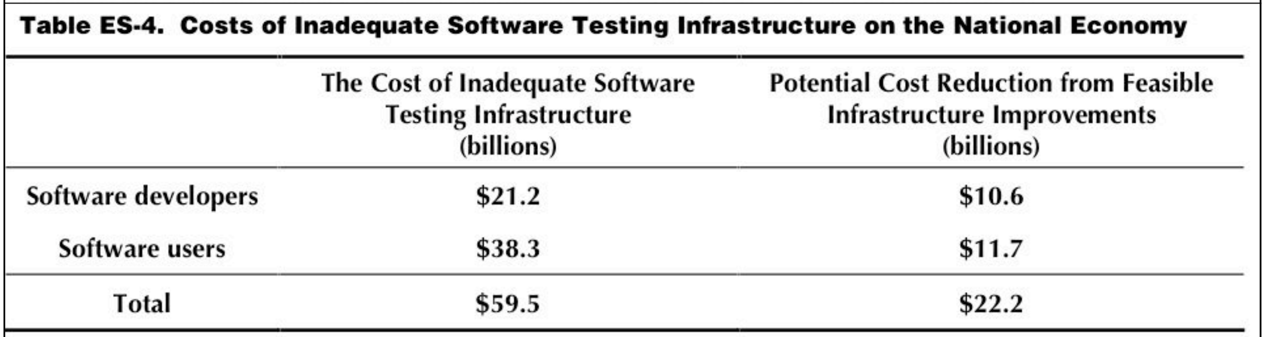
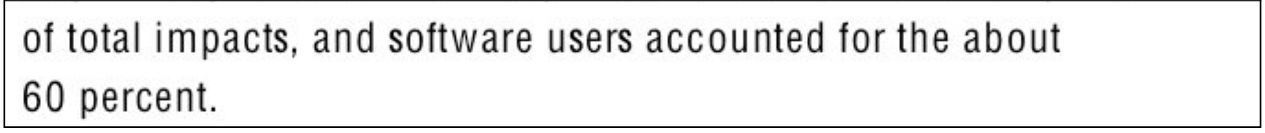


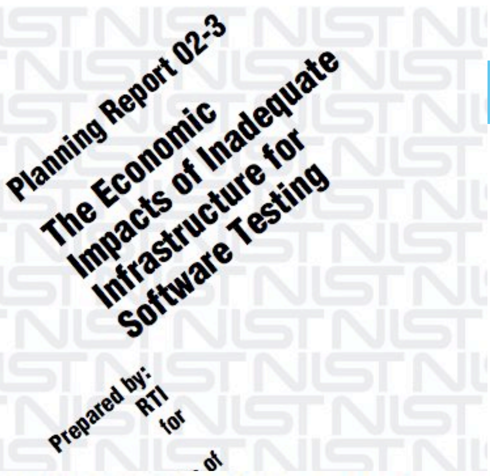
Table ES-4. Costs of Inadequate Software Testing Infrastructure on the National Economy

	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?



Planning Report 02-3

The Economic Impacts of Inadequate Infrastructure for Software Testing

Prepared by: RTI for of

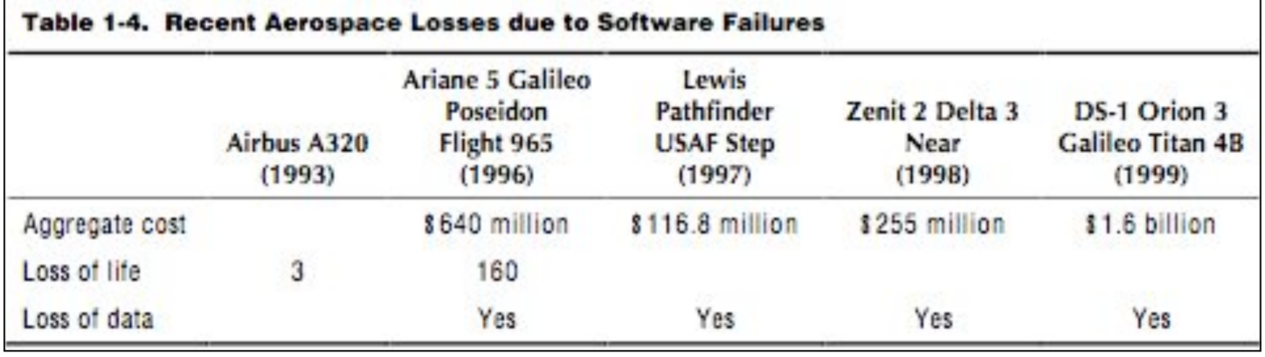
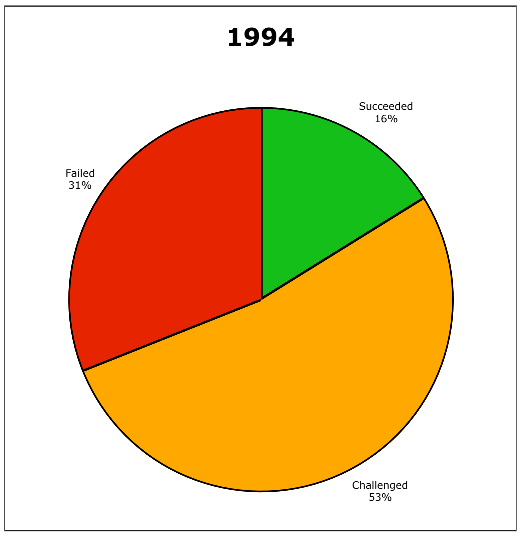
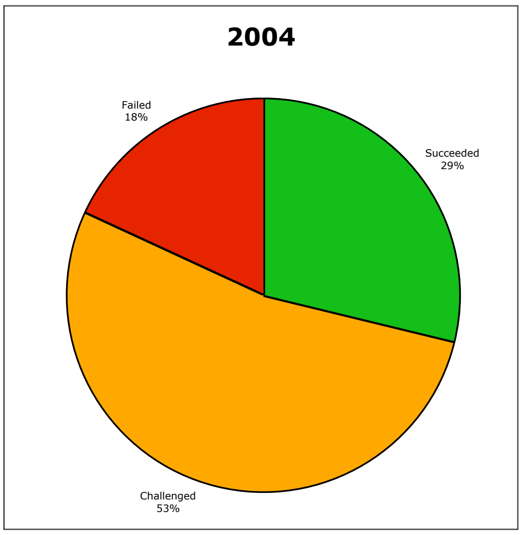


Table 1-4. Recent Aerospace Losses due to Software Failures

	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?





Standish Group, *"The CHAOS Report"*

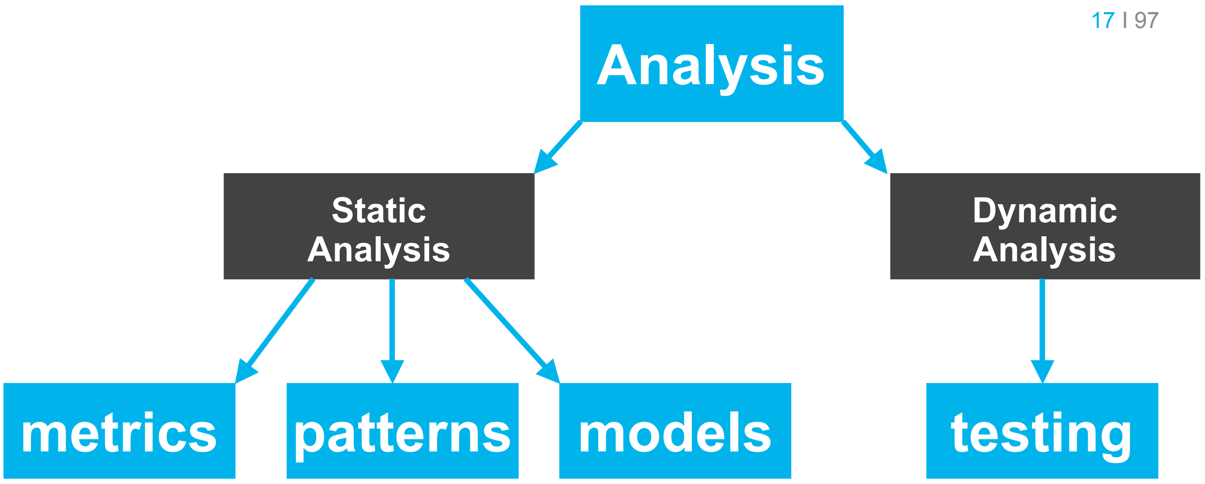
So

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

⇒

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

Structure of the lectures



TESTING

Testing

Kinds

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

Ways

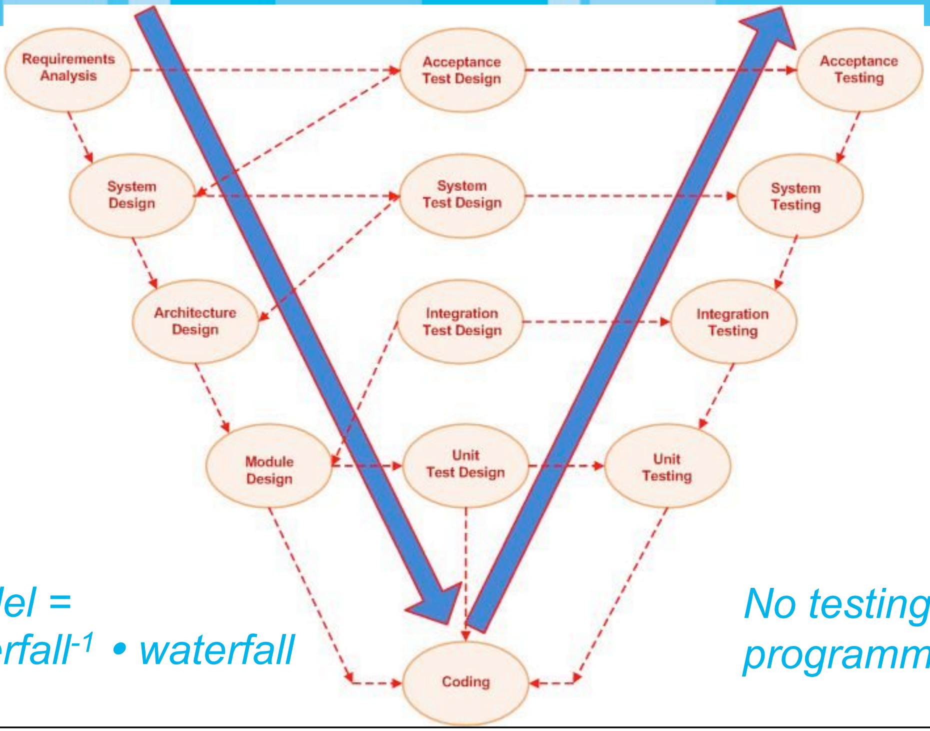
- Manual
- Automated
- Randomized
- Independent
- User
- Developer

With

- Plans
- Harness
- Data
- Method
- Frameworks

?

Testing V-model



V-model = waterfall^{-1} • waterfall

No testing while programming!

Testing
Eliminate waste

Software Improvement Group

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?

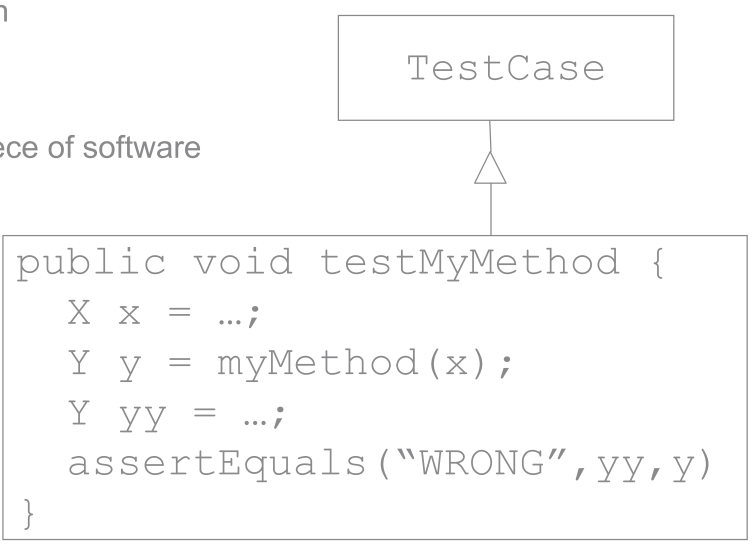
Software Improvement Group

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



```
public void testMyMethod {
  X x = …;
  Y y = myMethod(x);
  Y yy = …;
  assertEquals(“WRONG”,yy,y)
}
```

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

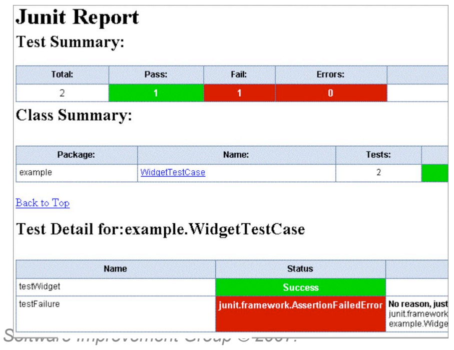
Stage 1
No unit testing

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

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

CLOVER
The industry standard in code coverage just got seriously better

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

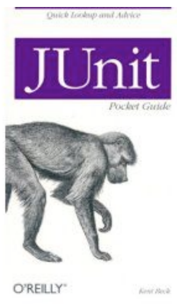
Stage 3
Coverage, not approaching 100%

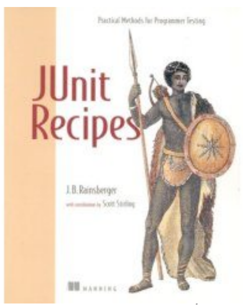
Excuses

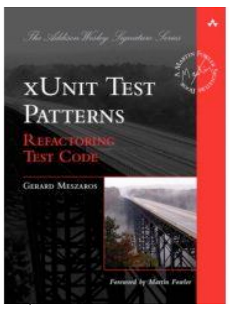
- “we test our libraries thoroughly, that effects more customers”

Actions:

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






The Addison-Wesley Signature Series
xUnit Test Patterns
Refactoring Test Code
Gerard Meszaros
Foreword by Martin Fowler

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

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

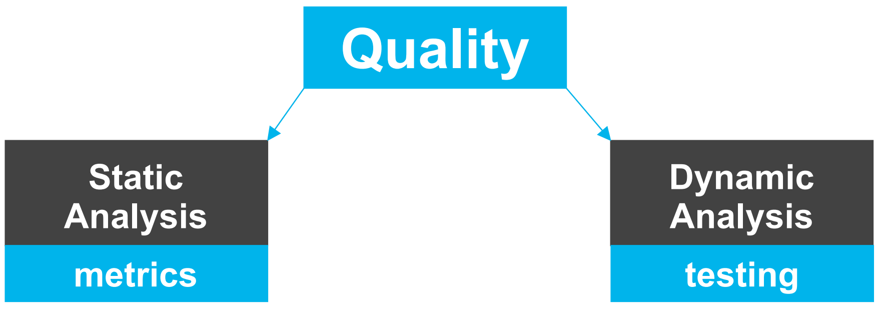
Test generation

Model-driven testing

Fault-injection

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

Projects



"interpretation of source code clusters"

"evaluation of QuickCheck solutions for Java"

Evaluate QuickCheck tools for Java

QuickCheck

- Randomized testing
- Specify properties
- QuickCheck tests each property with 100 randomly generated cases

Problem

- Originally for Haskell, now also for Erlang
- Several initiatives to develop QuickCheck for Java

Question

- Which QuickCheck for Java is the best?

Fun

- Find bugs in our programs, and get rewarded for it!

Is testing un-cool?

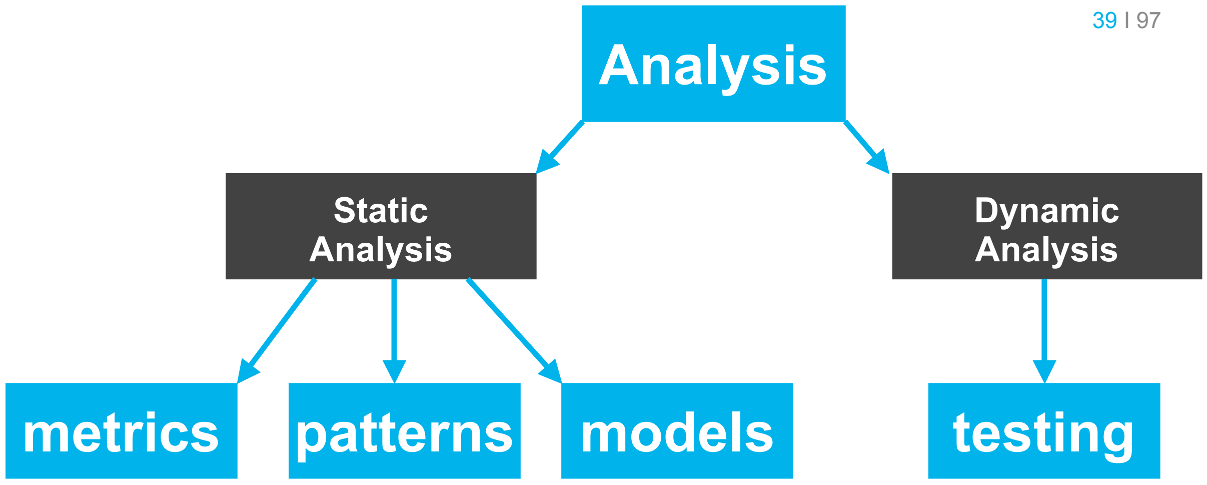
Edsger Wybe Dijkstra (1930 - 2002)

- “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.”

Structure of the lectures



PATTERNS

Patterns

Coding style and coding standards

- E.g. layout, identifiers, method length, …

Secure coding guidelines

- E.g. SQL injection, stack trace visibility

Bug patterns

- E.g. null pointer dereferencing, bounds checking

Code smells

- E.g. “god class”, “greedy class”, ..

Patterns
Style and standards

Checking coding style and coding standards

- Layout rules (boring)
- Identifier conventions
- Length of methods
- Depth of conditionals

Aim

- Consistency across different developers
- Ensure maintainability

Tools

- E.g. CheckStyle, PMD, …
- Integrated into IDE, into nightly build
- Can be customized

Patterns
Secure coding

Checking secure coding guidelines

- SQL injection attack
- Storing and sending passwords
- Stack-trace leaking
- Cross-site scripting

Aim

- Ensure security
- Security = Confidentiality + Integrity + Availability

Tools

- E.g. Fortify

Patterns
Bugs

Detecting bug patterns

- Null-dereferencing
- Lack of array bounds checking
- Buffer overflow

Aim

- Correctness
- Compensate for weak type checks

Tools:

- e.g. FindBugs
- Esp. for C, C++