Software Testing

Credits:
- IPL (Cantata++)
- Rick Mercer; Franklin, Beedle & Associates
- Satish Mishra; HU Berlin
- Hyoung Hong; Concordia University
- Pressman

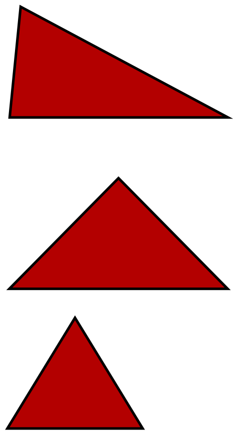
Instructor: Peter Baumann

email: p.baumann@jacobs-university.de
tel: -3178
office: room 88, Research 1

"Hey, it compiles – let's ship it!"

Test Your Testing!
[Myers 1982]

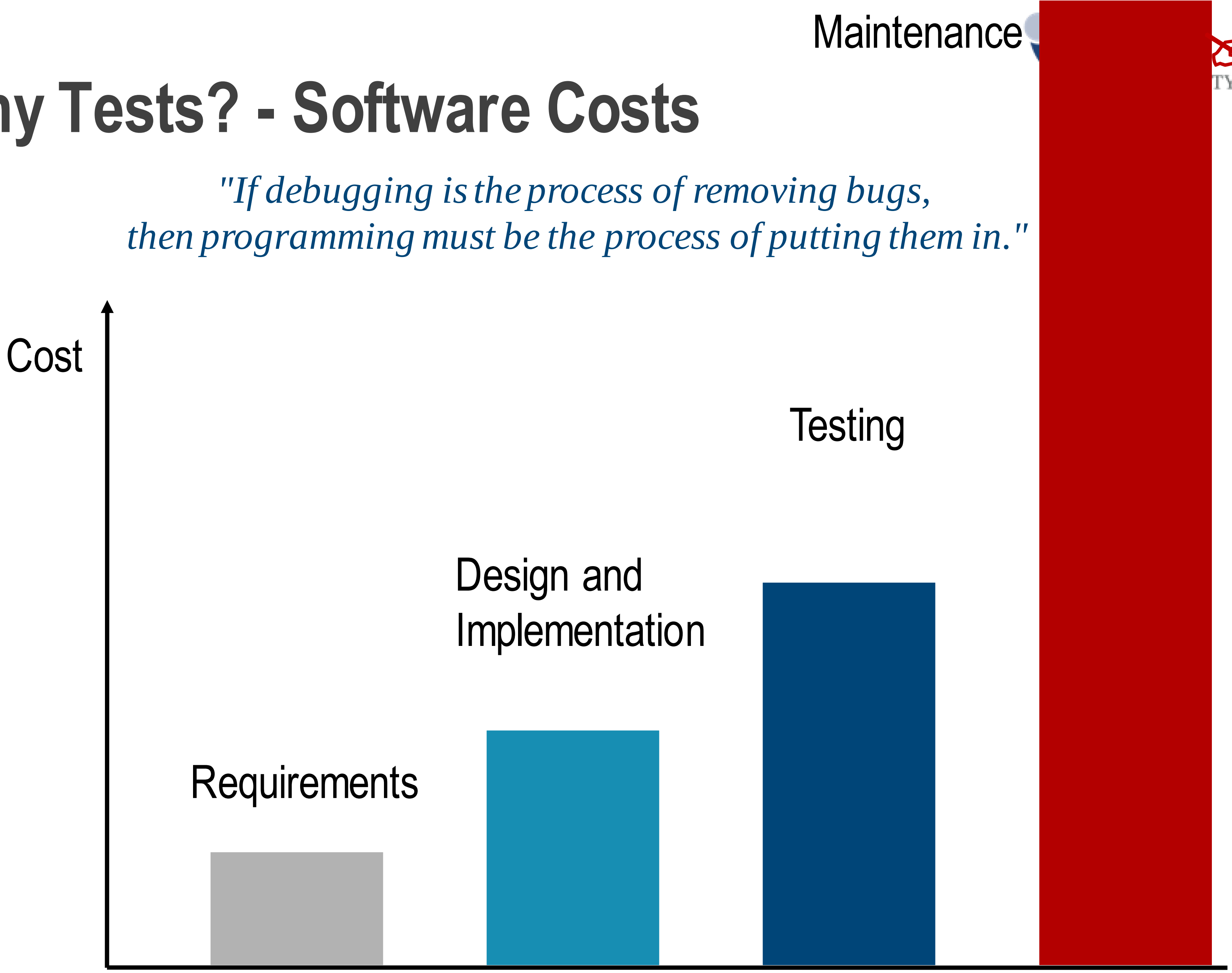
- Program reads 3 integers from cmd line, interprets as side lengths of a triangle
- Outputs triangle type:
  - Non-equilateral
  - Equilateral
  - Isosceles



- ...test cases?

Why Tests? - Software Costs

"If debugging is the process of removing bugs, then programming must be the process of putting them in."



Some *Better-Test-Well* Applications

Nuclear Reactor Control - Thales

Train Control - Alcatel

Medical Systems – GE Medical

Cantata++ running under Symbian – Nokia Series 60

EFA Typhoon – BAe Systems

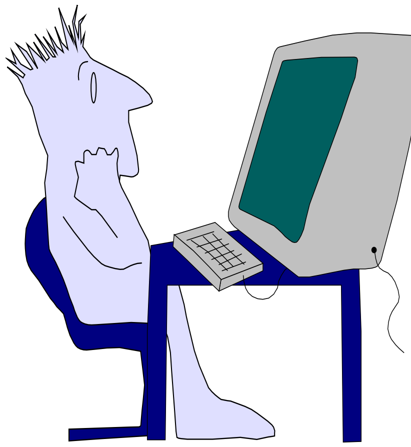
International Space Station – Dutch Space

Airbus A340 – Ultra Electronics

What Is Software Testing?

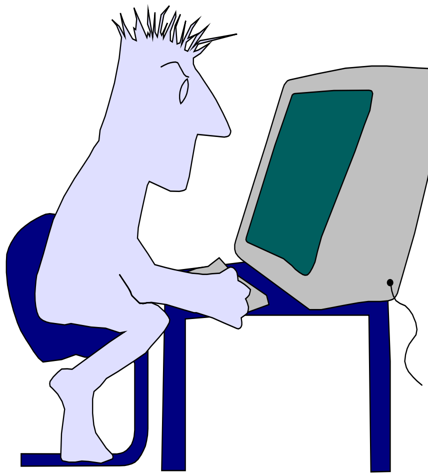
- Software Testing =
  process of exercising a program
  with the specific intent of finding errors
  prior to delivery to the end user.

Who Tests the Software?



developer

Understands the system
but will test "gently"
driven by "delivery"

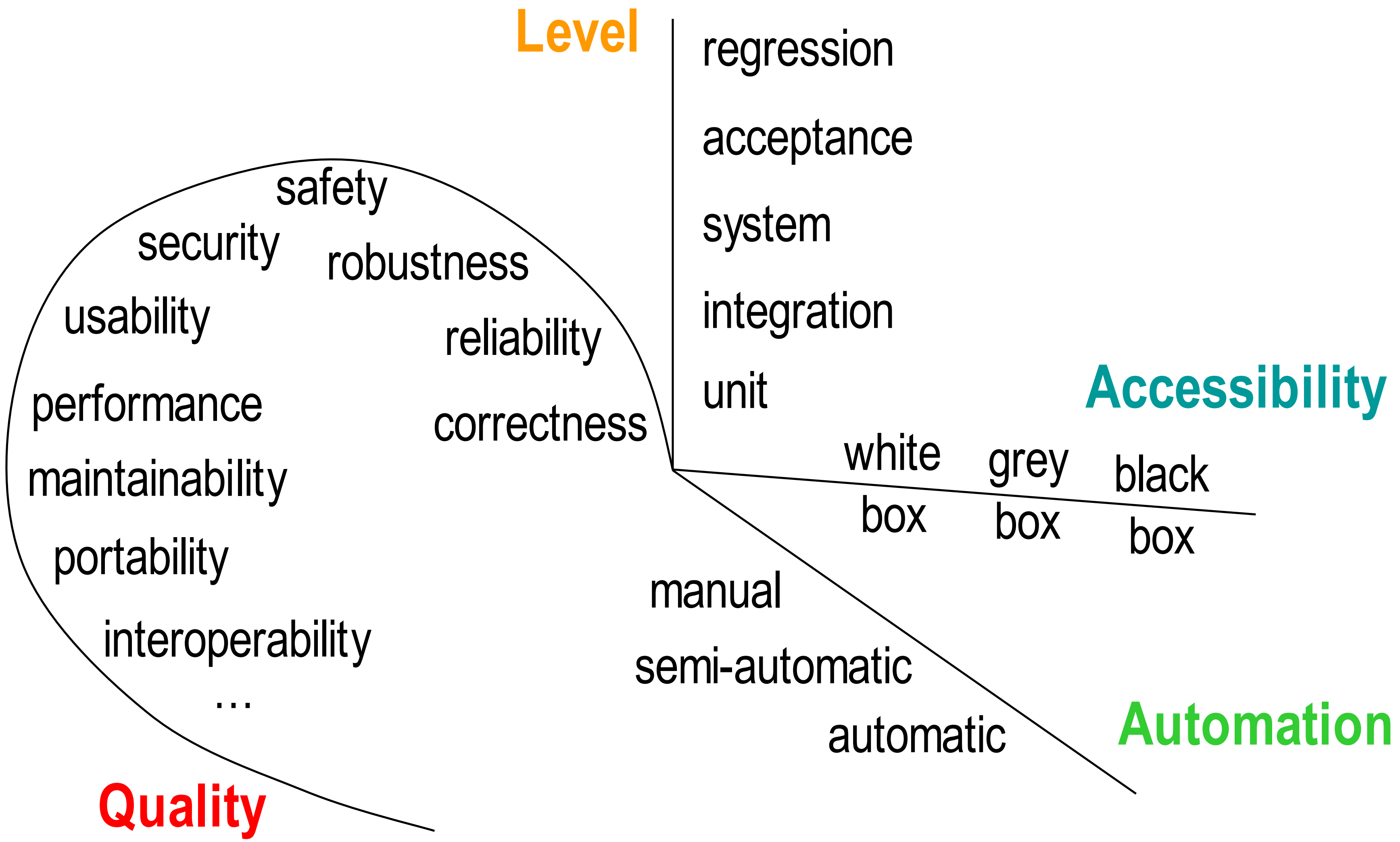


independent tester

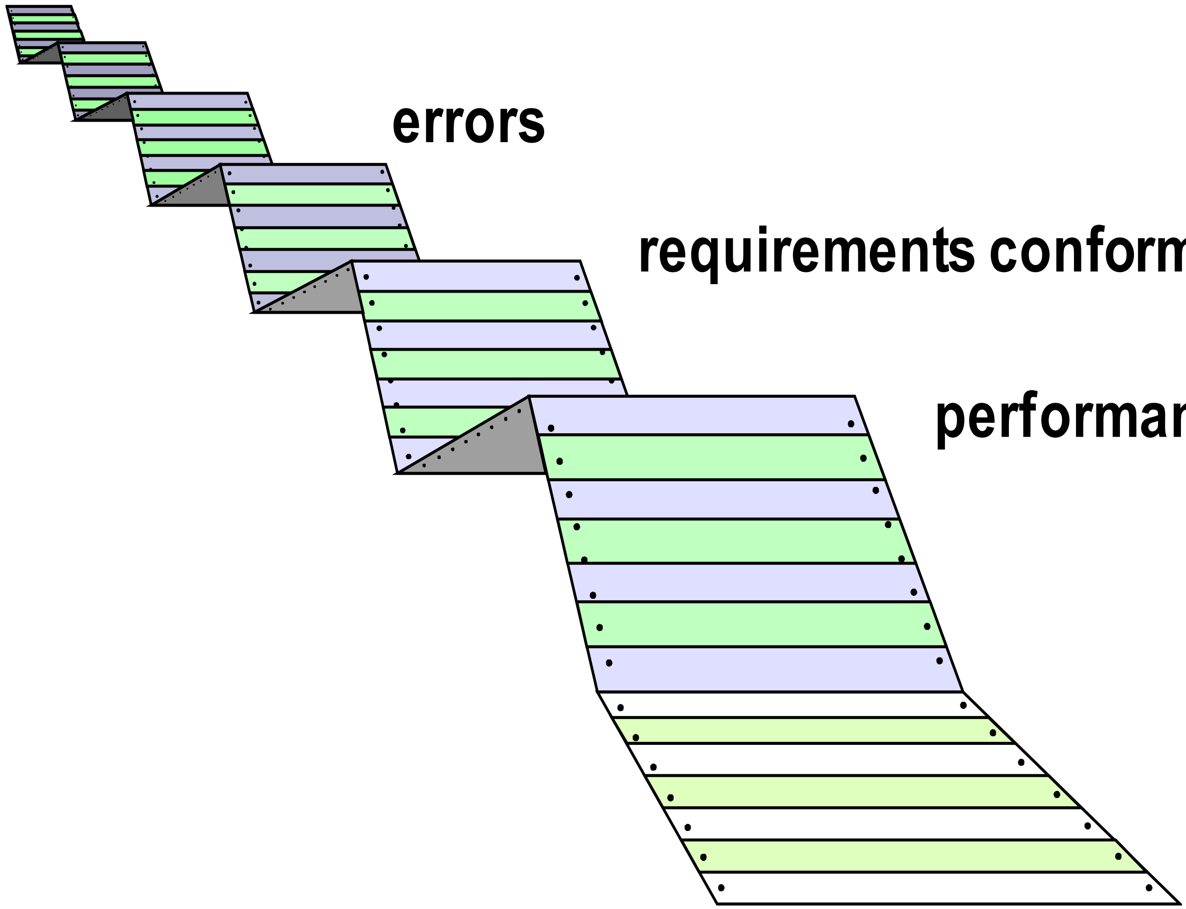
Must learn about the system
but will attempt to break it
driven by quality

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."
- Brian Kernighan

Test Feature Space

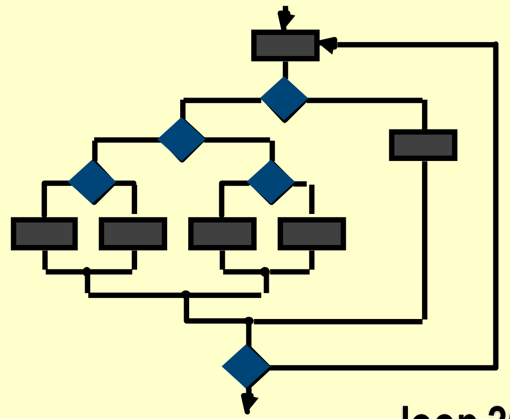


What Testing Shows



Equivalence Class Testing

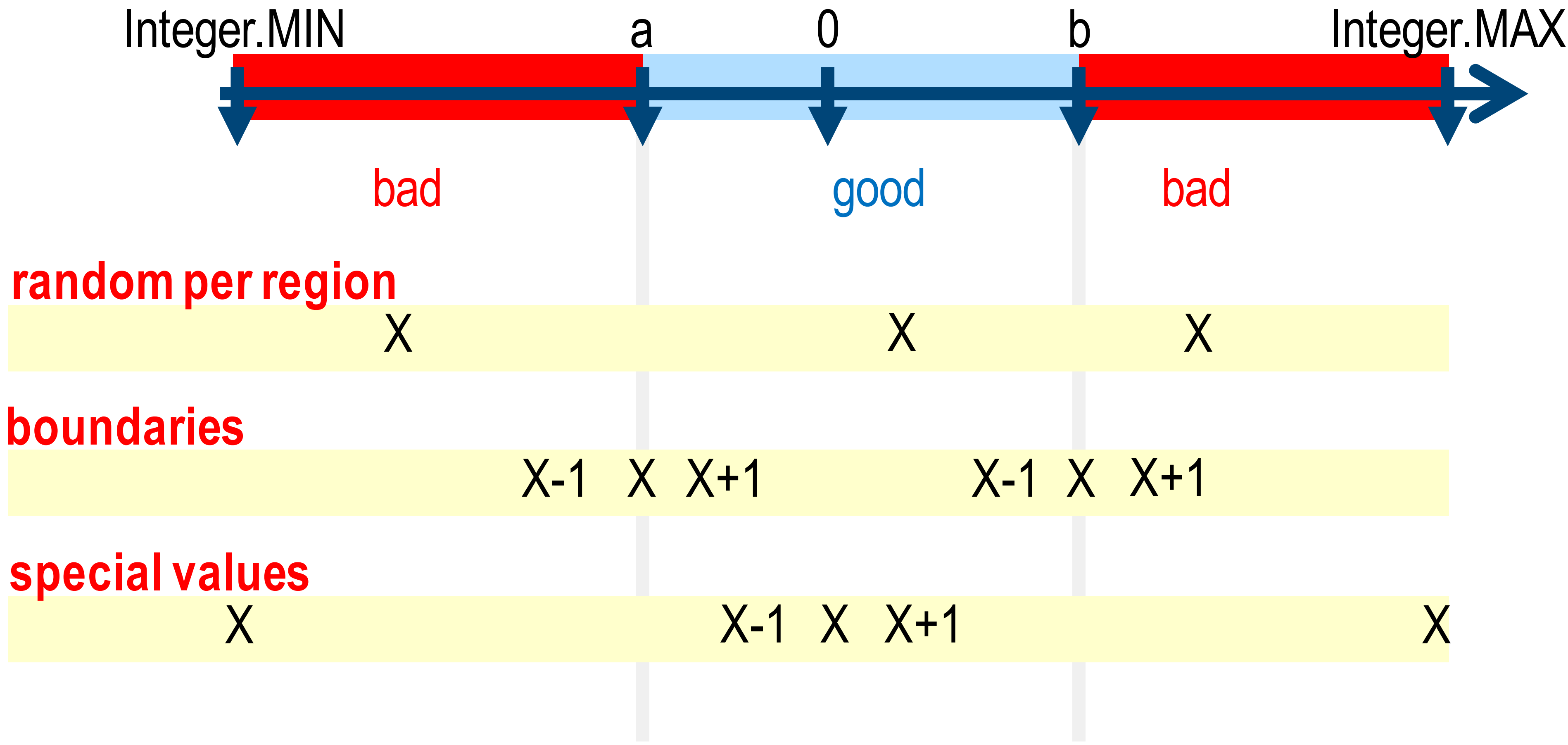
- Practically never can do exhaustive testing on input combinations
- How to find „good“ test cases?
  - Good = likely to produce an error
- Idea:
  build equivalence classes
  of test input situations,
  test one candidate per class



10^{14} possible paths;
1 test per millisecond
= 3,170 years to test completely

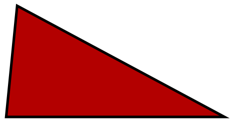
A Pragmatic Test Case Strategy

function f(int n) int with a<n<b:

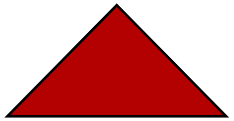


Test Your Testing, Reloaded

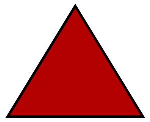
- Program reads 3 integers from cmd line, interprets as side lengths of a triangle
- Outputs triangle type:
  - Non-equilateral



  - Equilateral



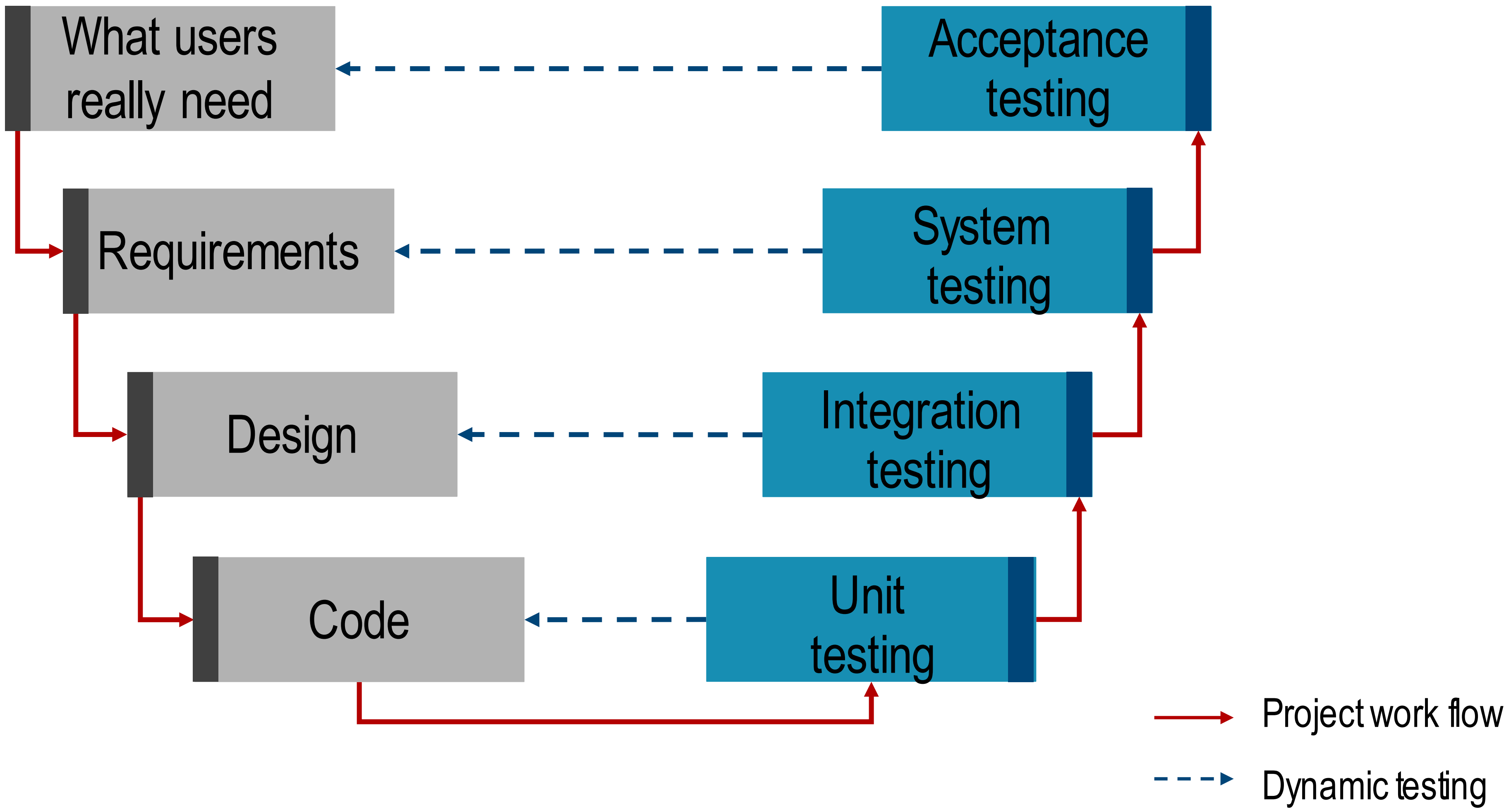
  - Isosceles



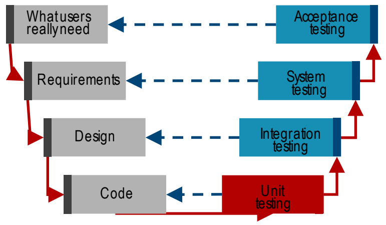
- ...test cases?

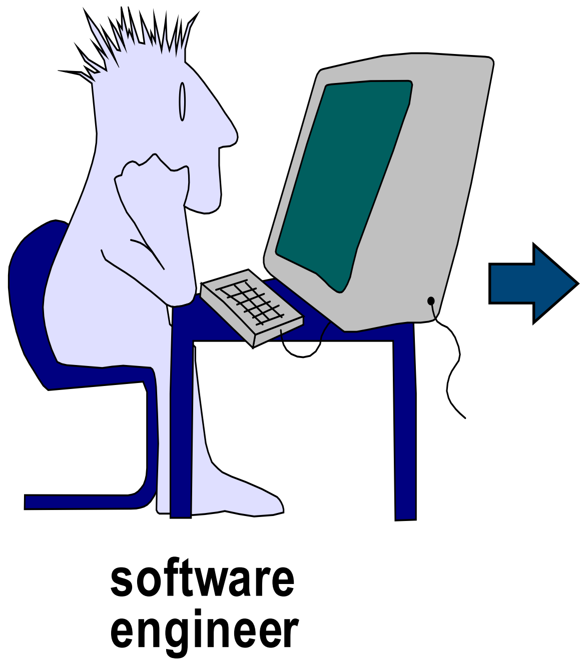
Testing & The Design Cycle

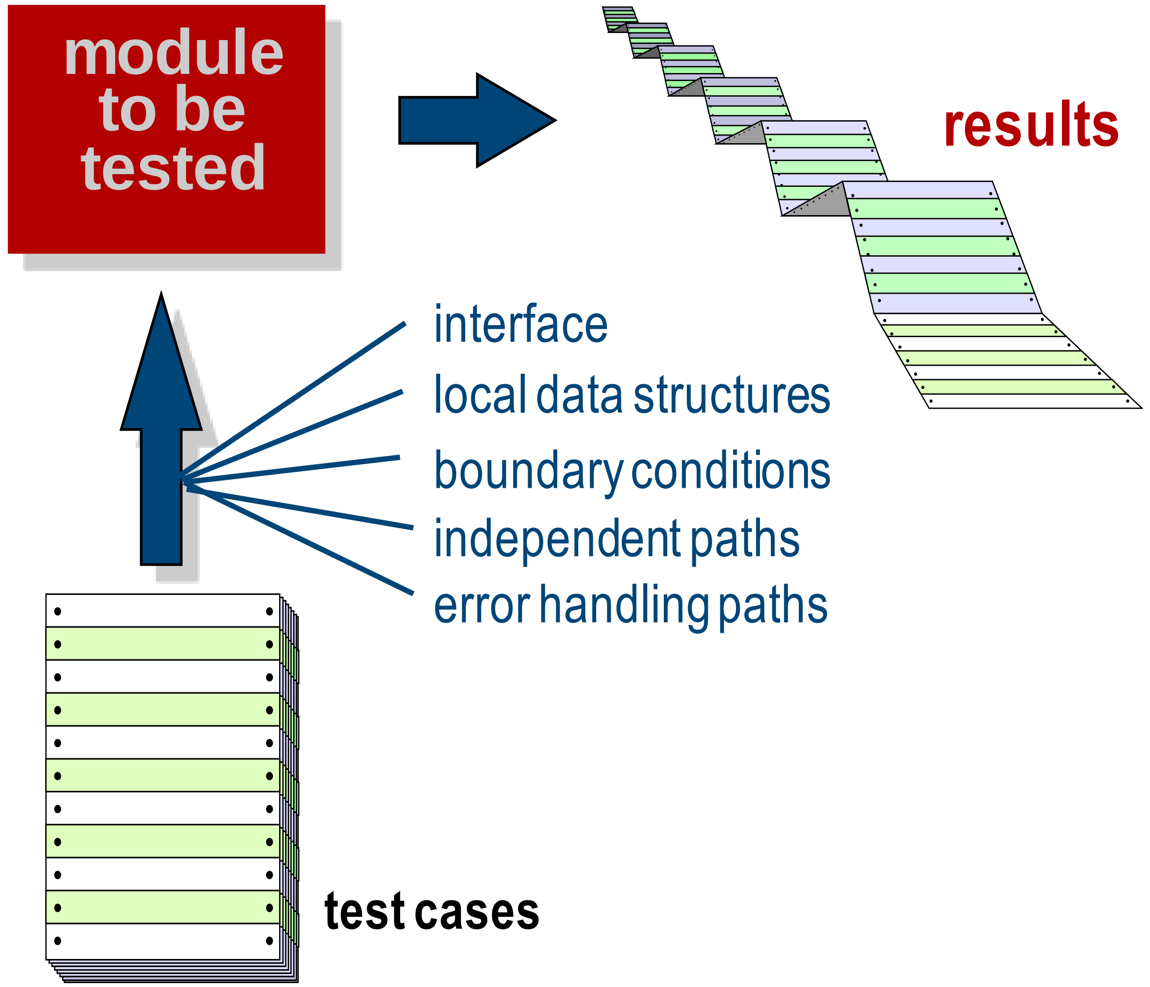
Missing: maintenance phase!



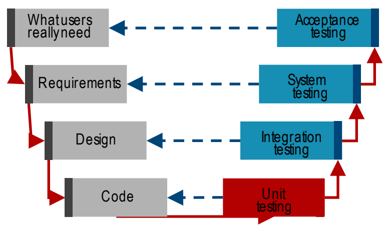
Unit Testing







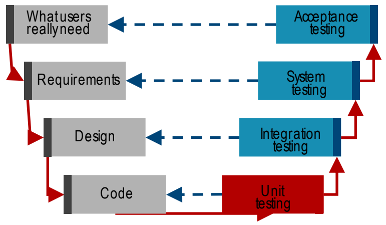
Unit Testing



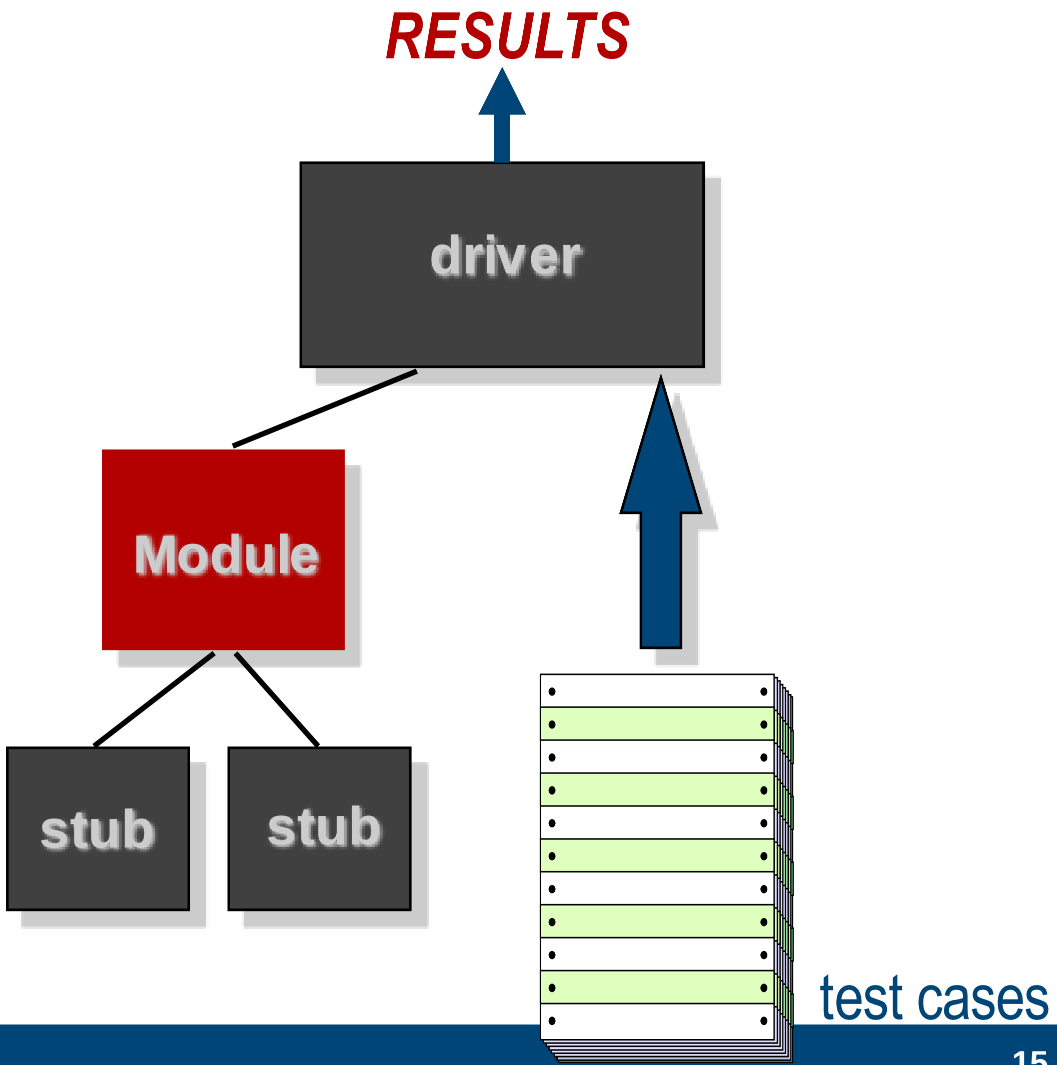
- Test unit = code that tests target
  - Usually one or more test module/class
  - In oo programs: target frequently one class
- Test case = test of an assertion ("design promise") or particular feature
  - "*writing to then deleting an item from an empty stack yields an empty stack*":

    isempty(pop(push(empty(), x)))

Unit Test Environment



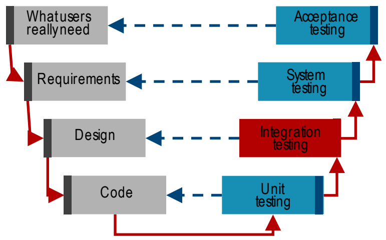
- Test driver
  = dummy environment
  for test class
- Test stub
  = dummy methods
  of classes used,
  but not available
- Some unit testing frameworks
  - C++: cppunit
  - Java: JUnit
  - server-side Java code (web apps!): Cactus
  - JavaScript: JSpec



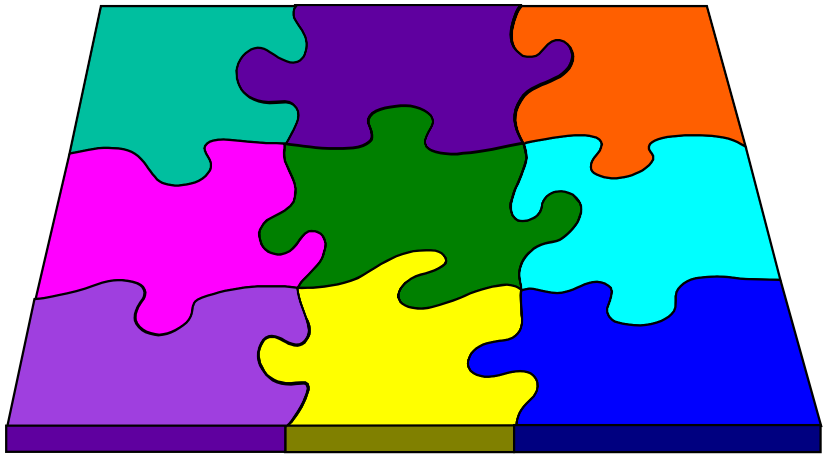
Test Software is Software!

- All quality aspects apply, such as:
- Code quality
- Documentation
  - „why is this test case important?“
- Automated handling via *make* etc.
- Appropriate structuring into directory hierarchies
  - Separate feature code & test code
- Example: rasdaman src tree

Integration Testing

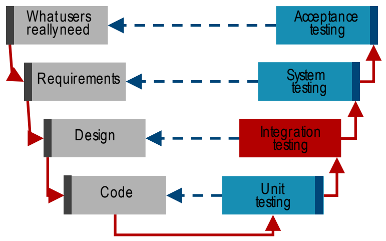


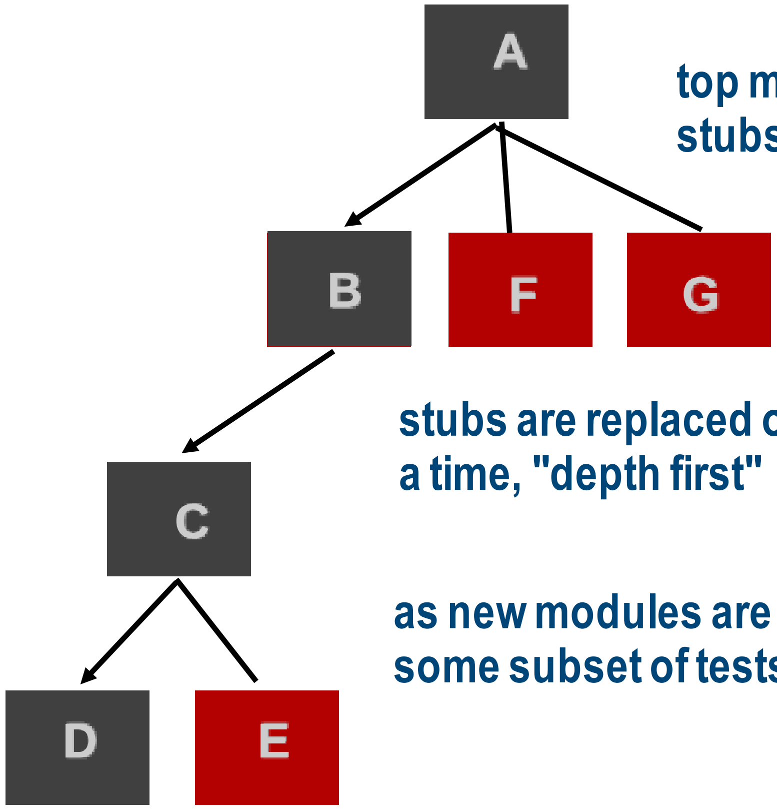
- Integration testing
  = test interactions among units
  - Import/export type compatibility
  - range errors
  - representation
  - *...and many more*



- Sample integration problems
  - F1 calls F2(char[] s) -- F1 assumes array of size 10, F2 assumes size 8
  - F1 calls F2(elapsed_time) -- F1 thinks in seconds, F2 thinks in milliseconds
- Strategies: Big-bang, incremental (top-down, bottom-up, sandwich)

Top-Down Integration



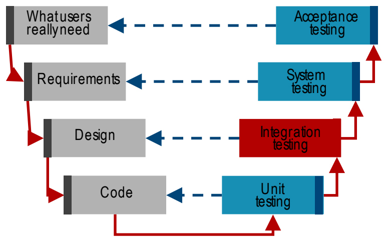


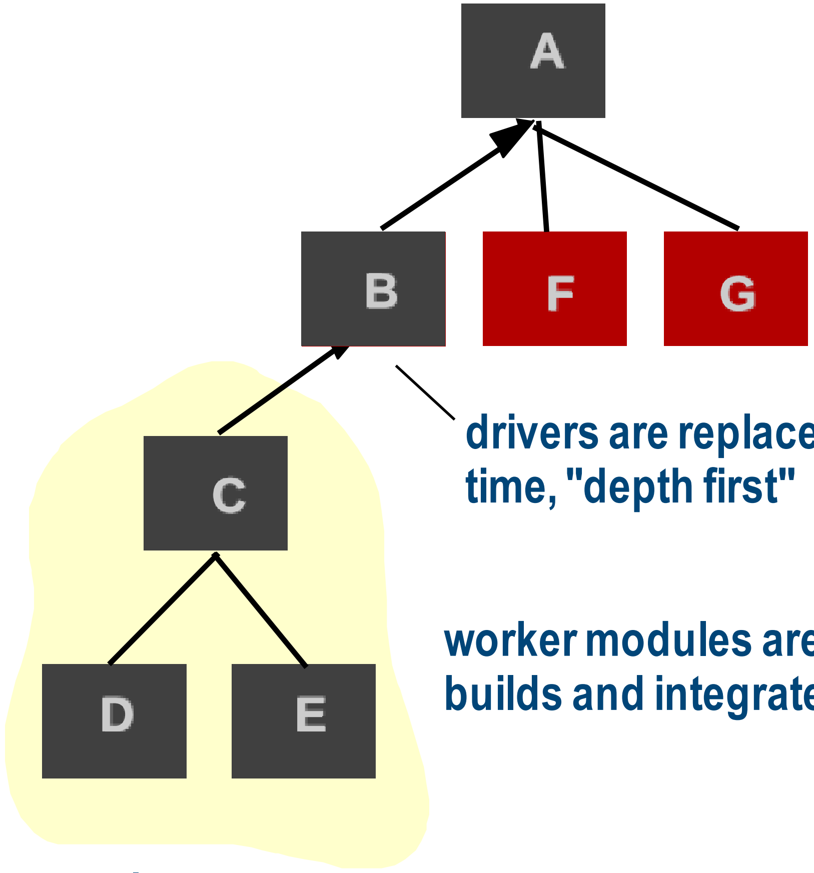
top module is tested with stubs

stubs are replaced one at a time, "depth first"

as new modules are integrated, some subset of tests is re-run

Bottom-Up Integration



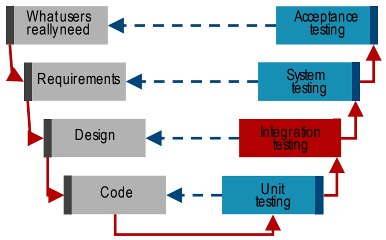


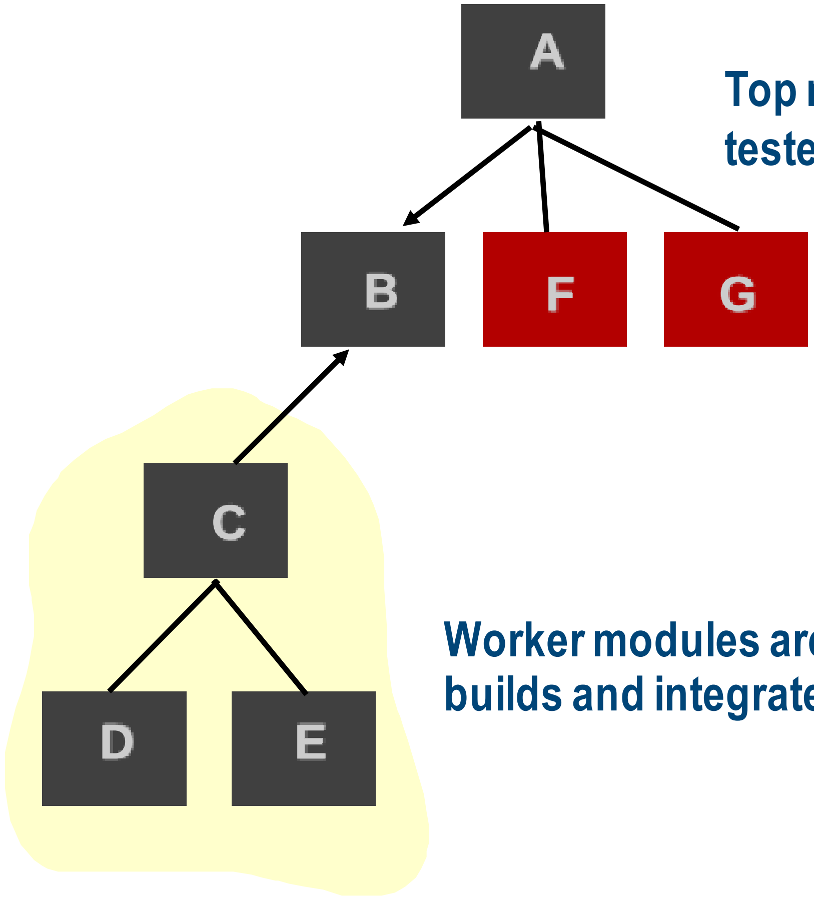
drivers are replaced one at a time, "depth first"

worker modules are grouped into builds and integrated

cluster

Sandwich Testing



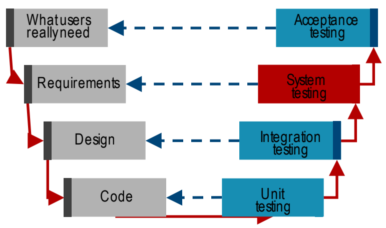


Top modules are tested with stubs

Worker modules are grouped into builds and integrated

cluster

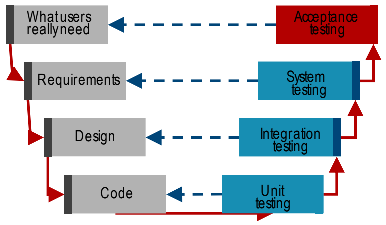
System Testing



- System testing =
  determine whether system meets requirements
  - = integrated hardware and software
- Focus on use & interaction of system functionalities
  - rather than details of implementations
- Should be carried out by a group independent of the code developers

- Alpha testing: end users at developer's site
- Beta testing: at end user site, w/o developer!

Acceptance Testing

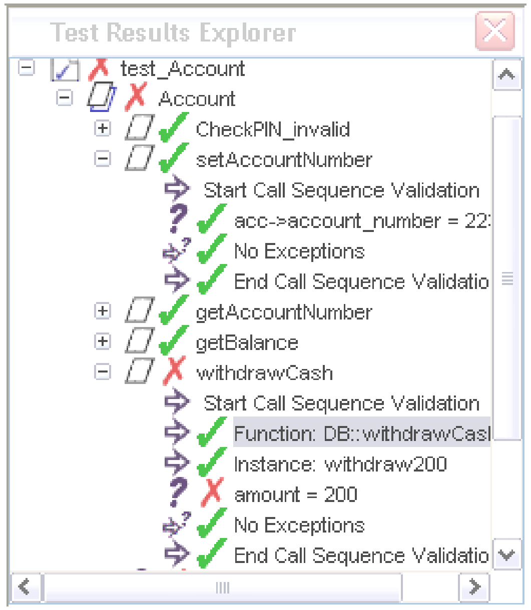


- Goal: Get approval from customer
  - try to structure it!
- be suresuresure that the demo works
- Customer may be tempted to demand more functionality when getting exposed to new system
  - Ideally: get test cases agreed already during analysis phase
  - …will not work in practice, customer will feel tied
  - At least: agree on schedule & criteria beforehand
- Best: prepare with stakeholders well in advance

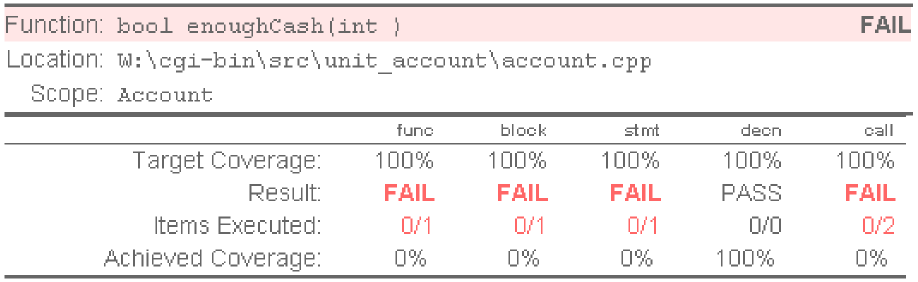
Testing Methods

static
dynamic
regression

- Static testing
  - Collects information about a software without executing it
  - *Reviews, walkthroughs, and inspections; static analysis; formal verification; documentation testing*
- Dynamic testing
  - Collects information about a software with executing it
  - Does the software behave correctly?
  - In both development and target environments?
  - *White-box vs. black-box testing; coverage analysis; memory leaks; performance profiling*
- Regression testing



```
Test Results Explorer
⊟ ✗ test_Account
  ⊟ ✗ Account
    ⊞ ✓ CheckPIN_invalid
    ⊟ ✓ setAccountNumber
          Start Call Sequence Validation
        ✓ acc->account_number = 22
        ✓ No Exceptions
        ✓ End Call Sequence Validatio
    ⊞ ✓ getAccountNumber
    ⊞ ✓ getBalance
    ⊟ ✗ withdrawCash
          Start Call Sequence Validation
        ✓ Function: DB::withdrawCasl
        ✓ Instance: withdraw200
        ✗ amount = 200
        ✓ No Exceptions
        ✓ End Call Sequence Validatio
```



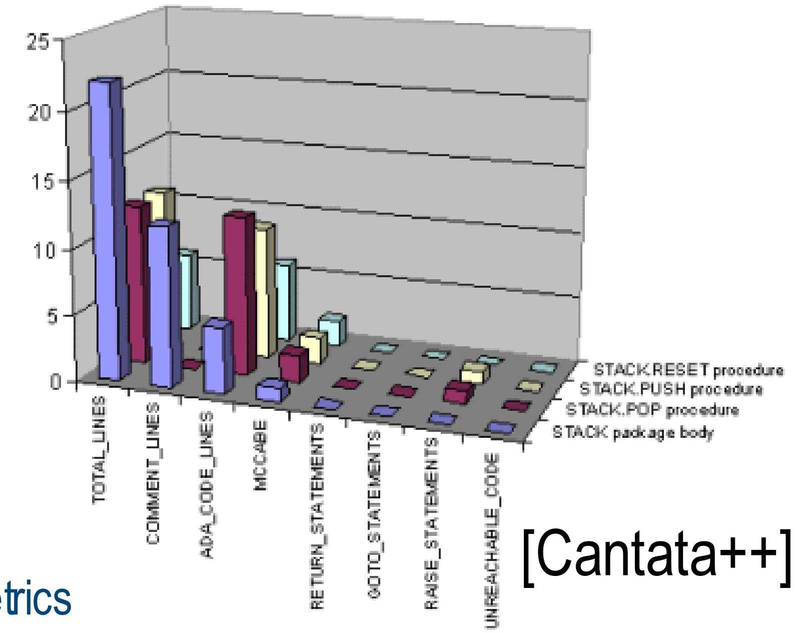
```
Function: bool enoughCash(int )                                   FAIL
Location: W:\cgi-bin\src\unit_account\account.cpp
   Scope: Account
```

	func	block	stmt	decn	call
Target Coverage:	100%	100%	100%	100%	100%
Result:	FAIL	FAIL	FAIL	PASS	FAIL
Items Executed:	0/1	0/1	0/1	0/0	0/2
Achieved Coverage:	0%	0%	0%	100%	0%

static
dynamic
regression

Static Analysis

- Control flow analysis and data flow analysis
  - Provide objective data, eg, for code reviews, project management, end of project statistics
  - Extensively used for compiler optimization and software engineering
- Examples of errors that can be found:
  - Unreachable statements
  - Variables used before initialization
  - Variables declared but never used
  - Possible array bound violations
- Extensive tool support for deriving metrics from source code
  - e.g. up to 300 source code metrics
  - Code construct counts, Complexity metrics, File metrics



[Cantata++]

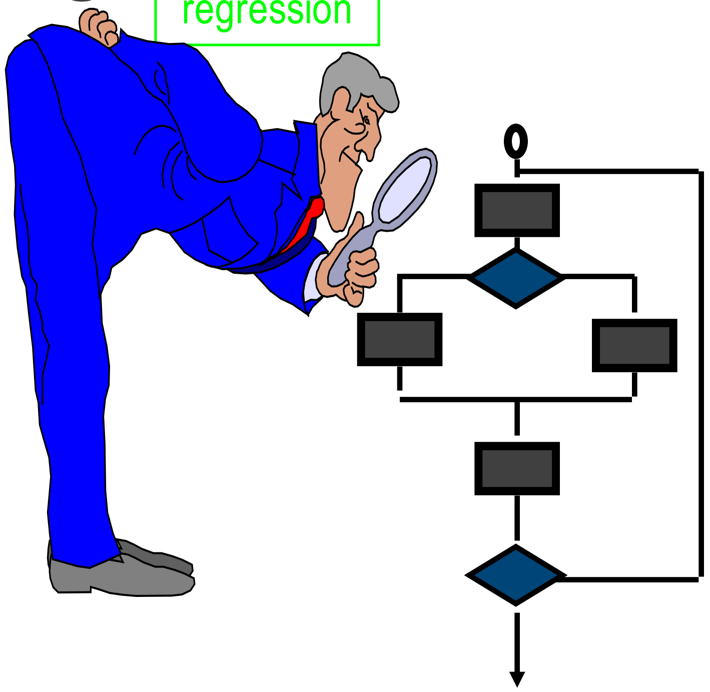
Formal Verification

static
dynamic
regression

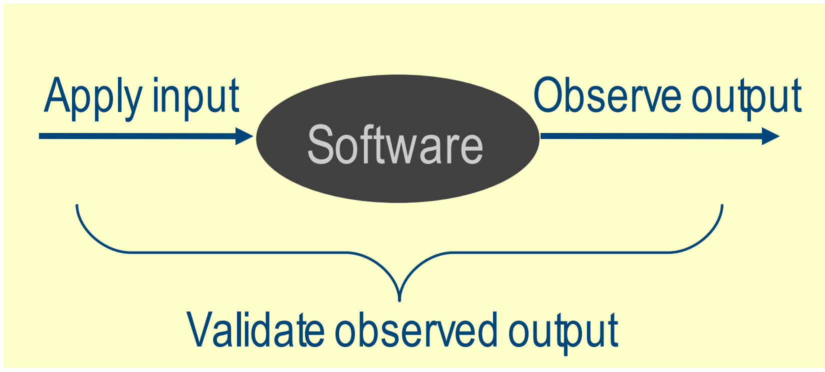
- Given a model of a program and a property, determine whether model satisfies property, based on mathematics
    - algebra, logic, …
    - *See earlier (invariants) and later!*
- Examples
    - Safety
        - *If the light for east-west is green, then the light for south-north should be red*
    - Liveness
        - *If a request occurs, there should be a response eventually in the future*

White-Box (Glass-Box) Testing

static
dynamic
regression



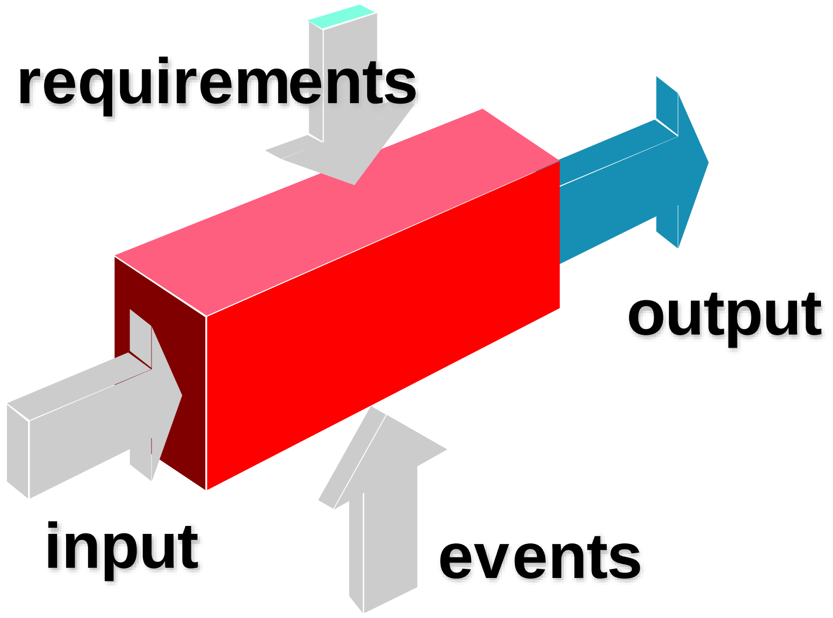
- Check that all statements & conditions have been executed at least once
- Look inside modules/classes
- Limitations
  - Cannot catch omission errors -- missing requirements?
  - Cannot provide test oracles -- expected output for an input?



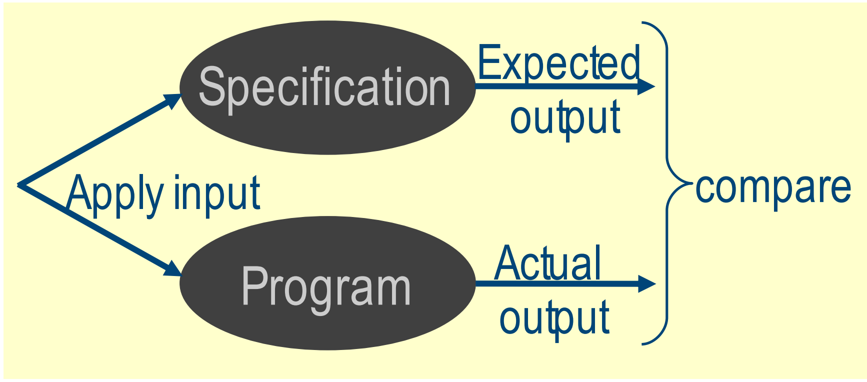
static
dynamic
regression

Black-Box = Spec-Based Testing

- No knowledge about code internals, relying only on interface spec



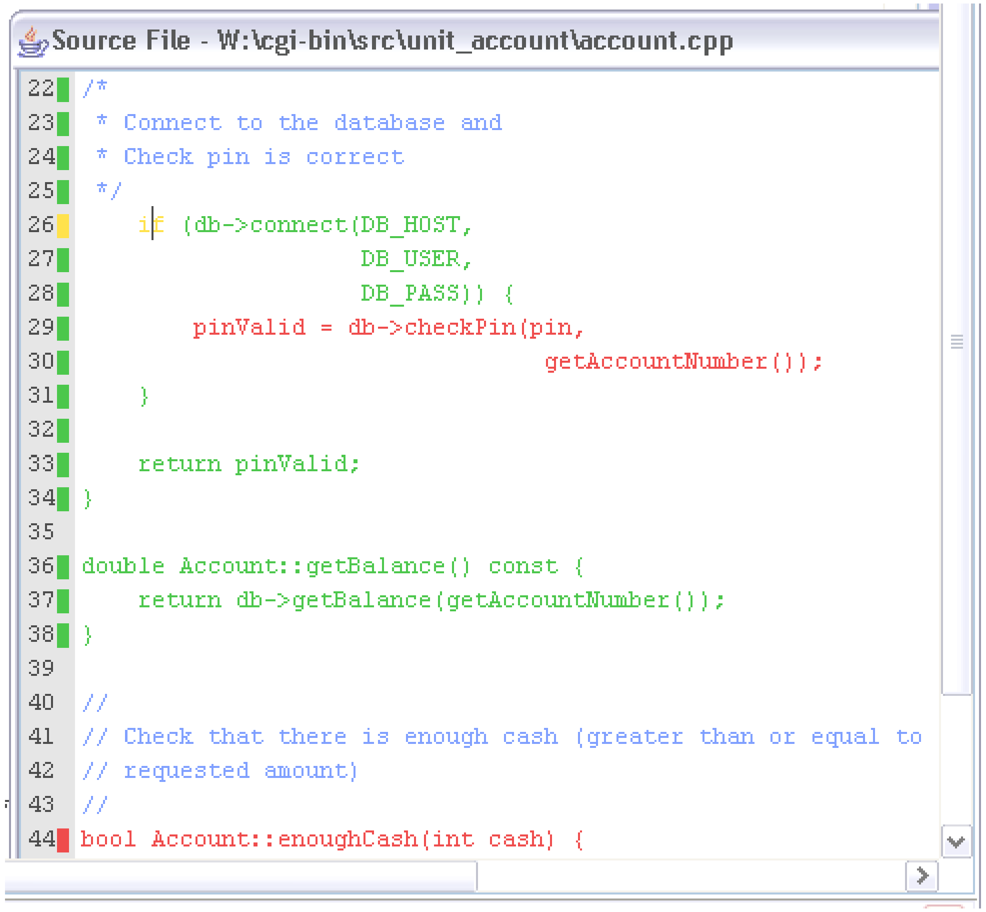
- Limitations
  - Specifications are not usually available
  - Many companies still have only code, there is no other document



static
dynamic
regression

Coverage Analysis

- Coverage analysis = measuring how much of the code has been exercised
  - identify unexecuted code structures
  - remove dead or unwanted code
  - add more test cases?
- Metrics include:
  - Entry points
  - Statements
  - Conditions (loops! ⤷)


Source File - W:\cgi-bin\src\unit_account\account.cpp

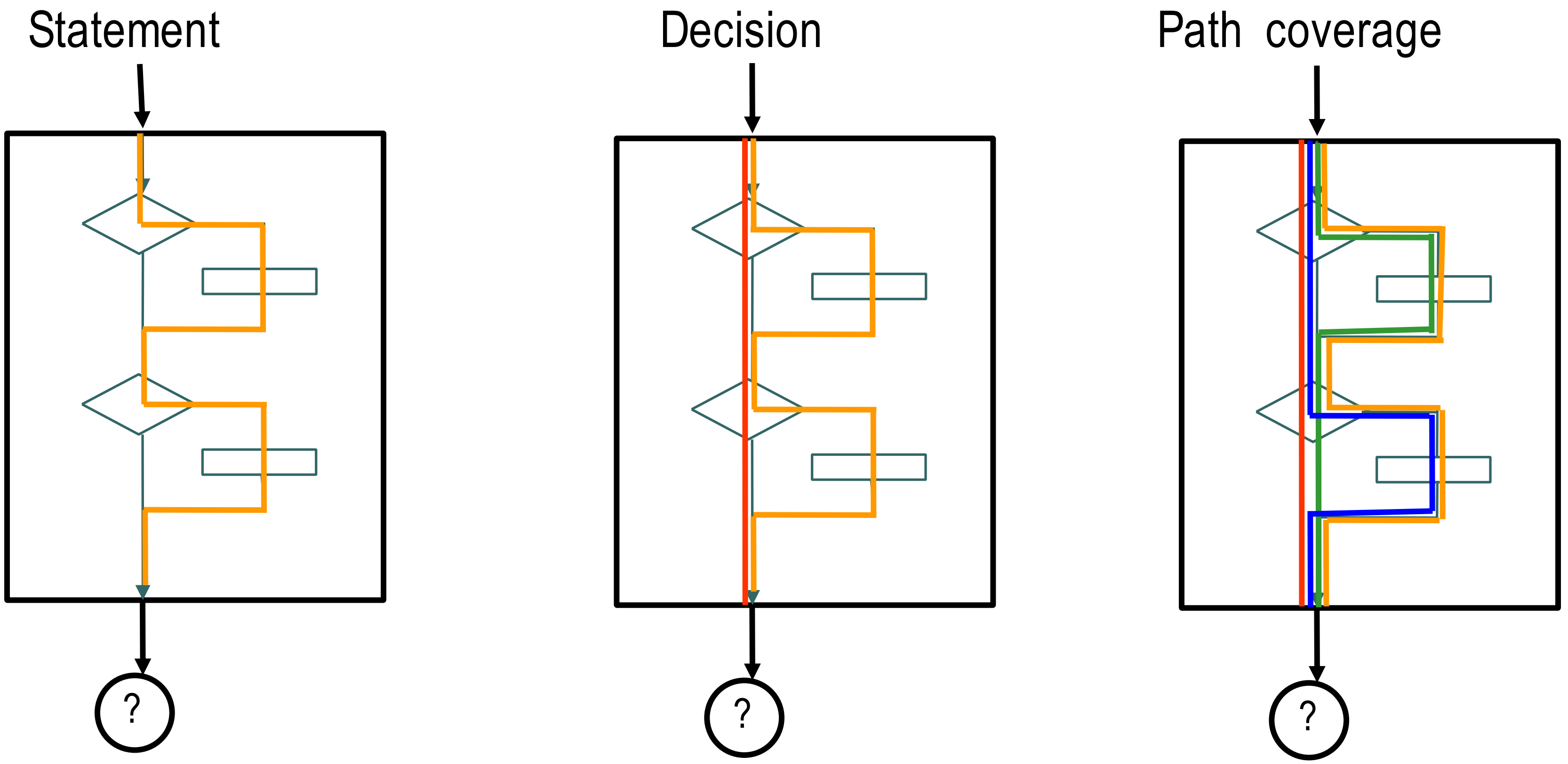
```
/*
 * Connect to the database and
 * Check pin is correct
 */
    if (db->connect(DB_HOST,
                    DB_USER,
                    DB_PASS)) {
        pinValid = db->checkPin(pin,
                                getAccountNumber());
    }

    return pinValid;
}

double Account::getBalance() const {
    return db->getBalance(getAccountNumber());
}

//
// Check that there is enough cash (greater than or equal to
// requested amount)
//
bool Account::enoughCash(int cash) {
```

Coverage Analysis: Metrics

static

dynamic

regression



# test cases?

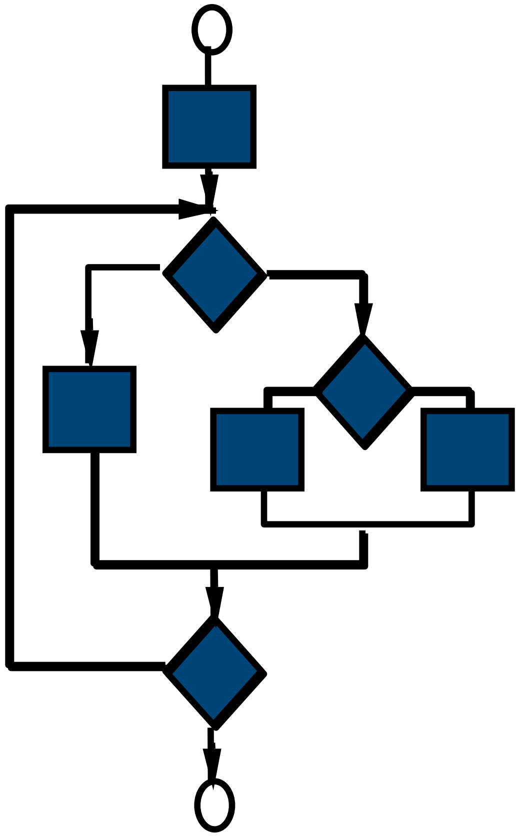
Path Testing

static

dynamic

regression

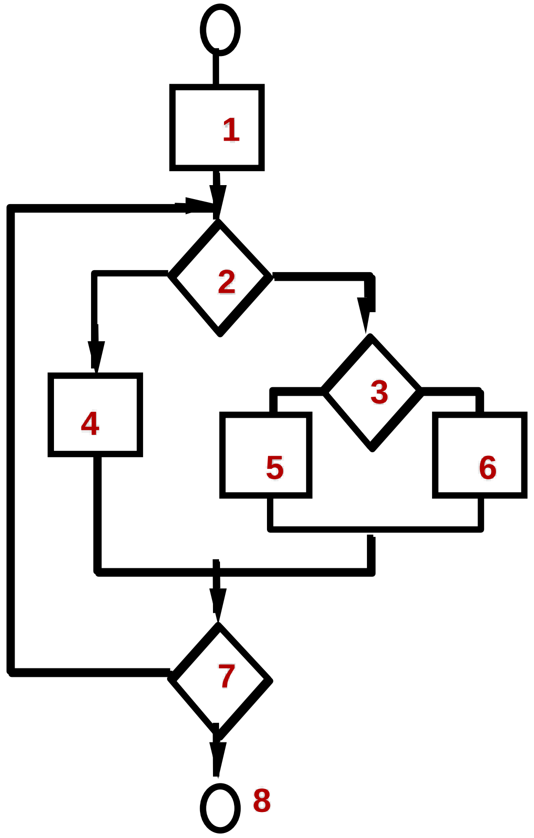
- cyclomatic complexity of flow graph:
- V(G) = number of simple decisions + 1
  - V(G) = number of enclosed areas + 1
- In this case, V(G) = ?



Path Testing

static
dynamic
regression

- derive independent paths: V(G) = 4 → four paths
  - Path 1: 1,2,3,6,7,8
  - Path 2: 1,2,3,5,7,8
  - Path 3: 1,2,4,7,8
  - Path 4: 1,2,4,7,2,4,...7,8
- derive test cases to exercise these paths



Terminology: Cx

What would you test?

- C0 = every instruction
- C1 = every branch
- C2, C3 ~= every condition once `true`, once `false`
  - Numbering historically grown, not systematic -- C1 & C2 not related!
- C4 = path coverage: every possible path taken

- Rule of thumb: 95% C0, 70% C1
  - C2, C3 IMHO add no value, C4 often impossible
- Concurrent systems? External component impact?

Example: DO-178B

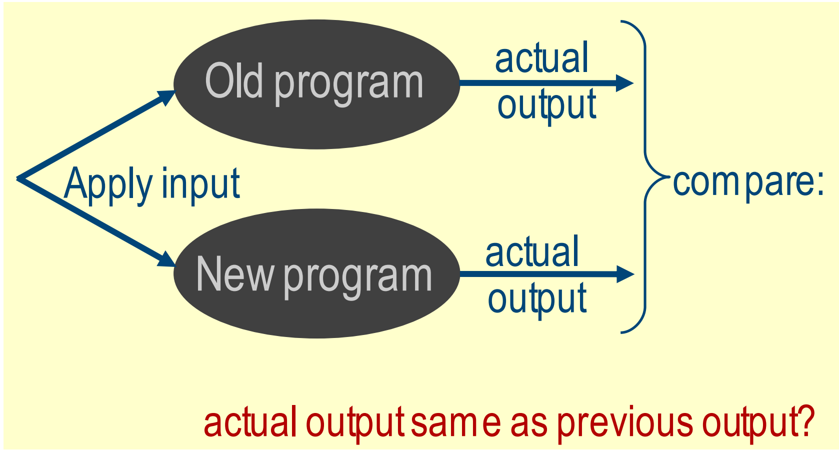
- FAA standard for requirements based testing & code coverage analysis
- Levels according to severity of consequences:

	…100% of:
Level A: catastrophic	*Modified cond. decision covg. + branch/decision + statement*
Level B: dangerous/severe	*Branch/decision + statement*
Level C: significant	*statement*
Level D: low impact	
Level E: no impact	

Regression Testing

static
dynamic
regression

- Testing in maintenance phase: How to test modified / new code?
  - Developing new tests = double work
  - Cost factor: Development : maintenance = 1:3
- Regression test
  = run tests, compare output
  to same test on previous code version
  - Diff on previous log output
  - easy automatic testing
- Limitations
  - Finds only deviations, cannot judge on error
  - Only finds new deviations
  - Only for fully automated tests



Test Organization

- Tests should be self-sustaining
  - create your own data,
  - ...and clean up
  - Expect nothing!
- Set up controlled enviroment
  - data sets, files, environment variables, system configuration, ...
  - excellent for repeatability of complex setup: virtual machines (eg, VMware box)
- Regression testing! ➯

Create Testable Software!

- Simplicity
  - Clear, easy to understand, following code standards
- Decomposability
  - Modules can be tested independently
- Controllability
  - States & variables can be controlled
  - tests can be automated and reproduced
- Observability
  - Make status queryable: toString()
  - Have class-internal checks & logging
- Stability
  - Recovers well from failures
- Operability
  - If well done right away, testing will be less blocked by errors found
- Understandability
  - All relevant information is documented, up-to-date, and available

Summary

- Pressman:
  - Think about what you see
  - Use tools to gain more insight
  - Create regression tests when fixing the bug
- Testing is hostile -- *„Make Test Like War!“*
  - be bad = imaginative on possible error situations
  - best be developed NOT by (but in communication with) coder
  - Common mistake: test only plausible input
    - OWASP, Snyk; OSS Fuzz: ~25,000 bugs in 375 OS tools

Summary (contd.)

- Objective test strategy should achieve
  "an acceptable level of confidence
  at an acceptable level of cost"
- Tests are integral part of the software
  - All quality statements apply!
  - ~40% of overall coding effort ok
- "*Testing is successful if the program fails*" – Goodenough & Gerhart
- "*Testers are customer advocates*" – n.n.