JACOBS
UNIVERSITY

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

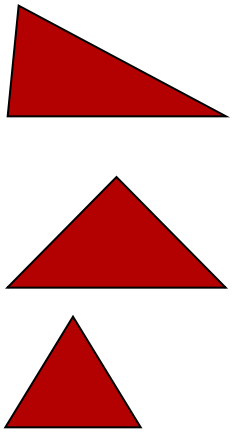
office: room 88, Research 1

"Hey, it compiles – let's ship it!"

Test Your Testing! [Myers 1982]

JACOBS UNIVERSITY

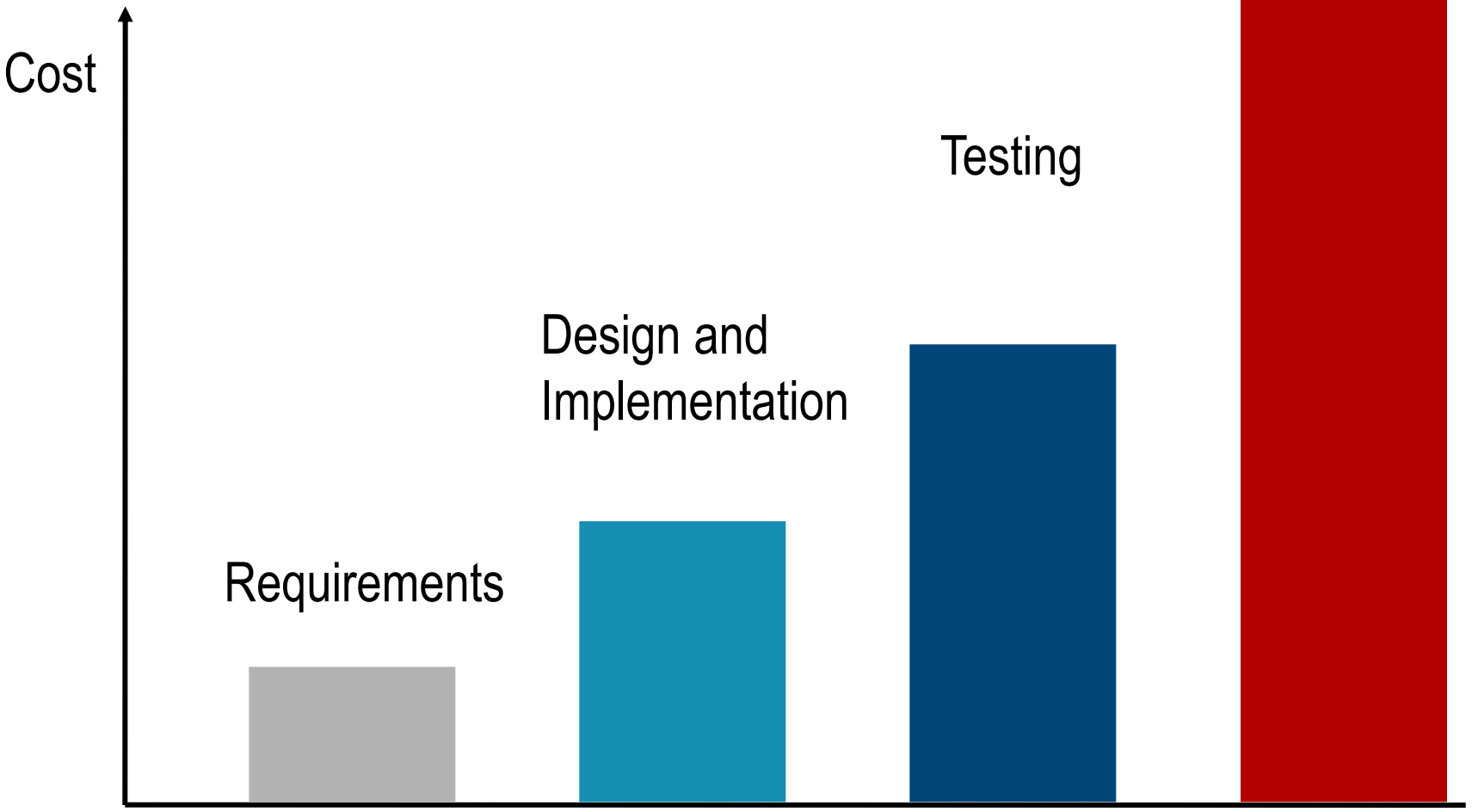
- Program reads 3 integers from cmd line, interprets as side lengths of a triangle
- Outputs triangle type:
  - Non-equilateral
  - Equilateral
  - Isosceles



- ...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

JACOBS UNIVERSITY

Nuclear Reactor Control - Thales

Train Control - Alcatel

EFA Typhoon – BAe Systems

Medical Systems – GE Medical

International Space Station – Dutch Space

Cantata++ running under Symbian – Nokia Series 60

Airbus A340 – Ultra Electronics

What Is Software Testing?

JACOBS UNIVERSITY

- 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?

JACOBS UNIVERSITY

developer

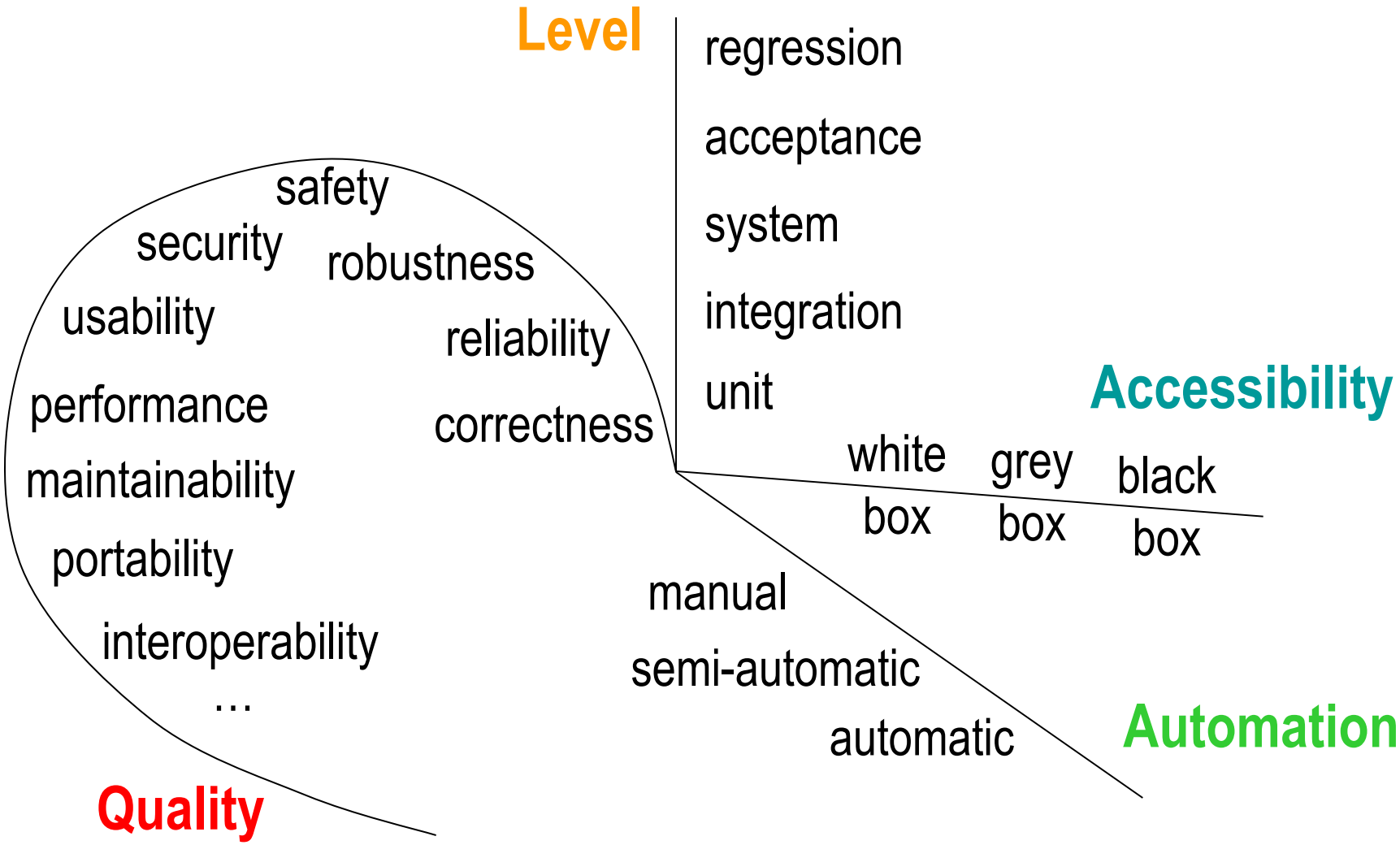
Understands the system
but will test "gently"
driven by "delivery"

independent tester

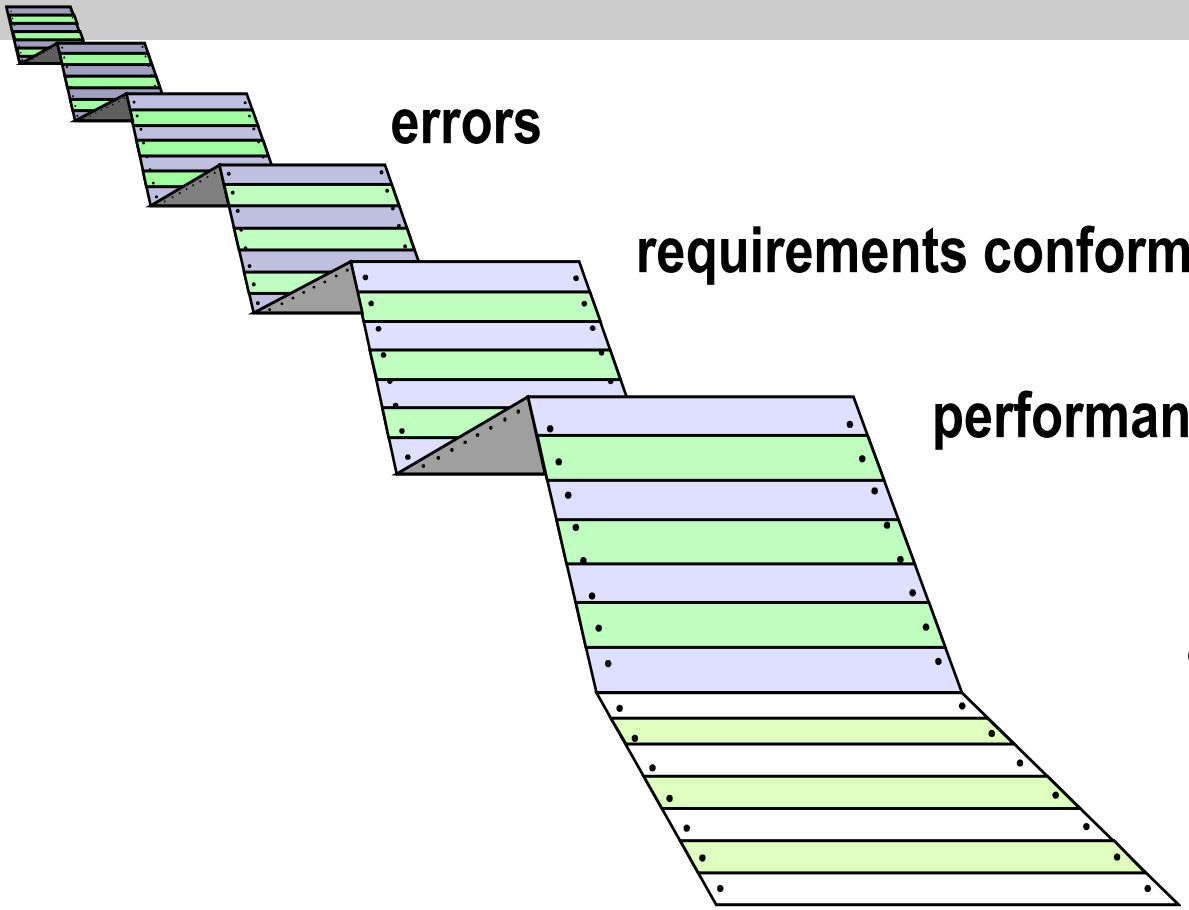
Must learn about the system
but will attempt to break it
and is driven by quality

*"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



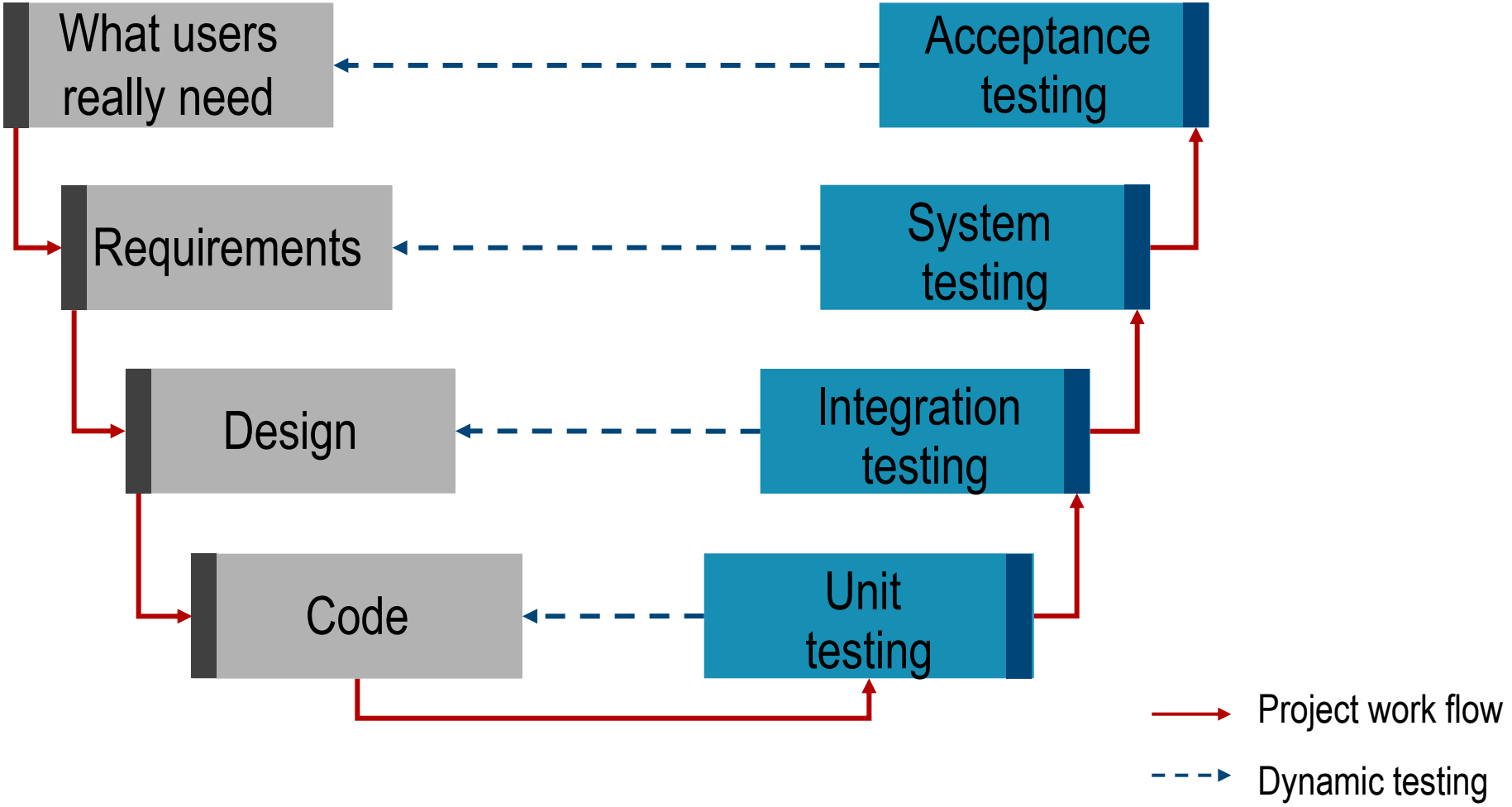
What Testing Shows



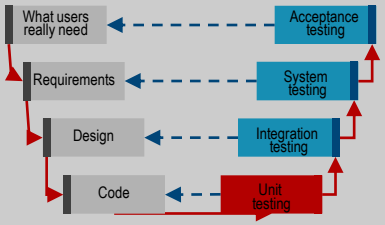
Testing & The Design Cycle

Missing: maintenance phase!

JACOBS UNIVERSITY



Unit Testing

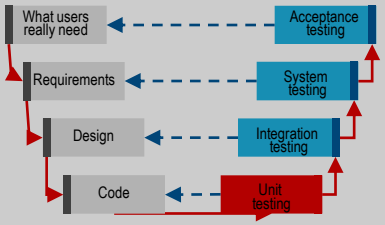


JACOBS UNIVERSITY

- 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)))

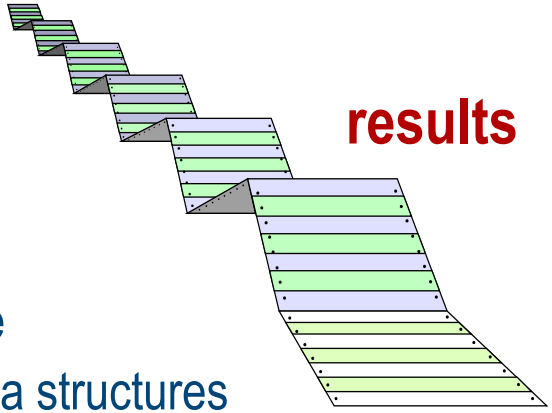
Unit Testing



JACOBS UNIVERSITY

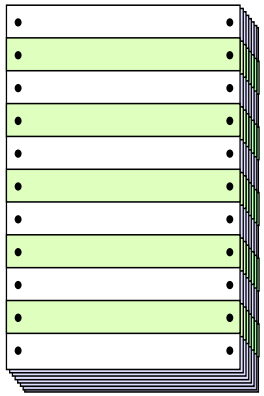
software engineer

module to be tested



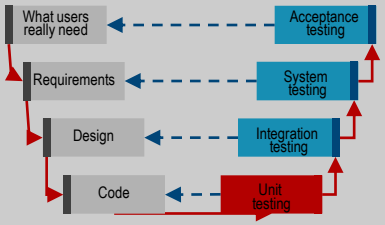
results

interface
local data structures
boundary conditions
independent paths
error handling paths



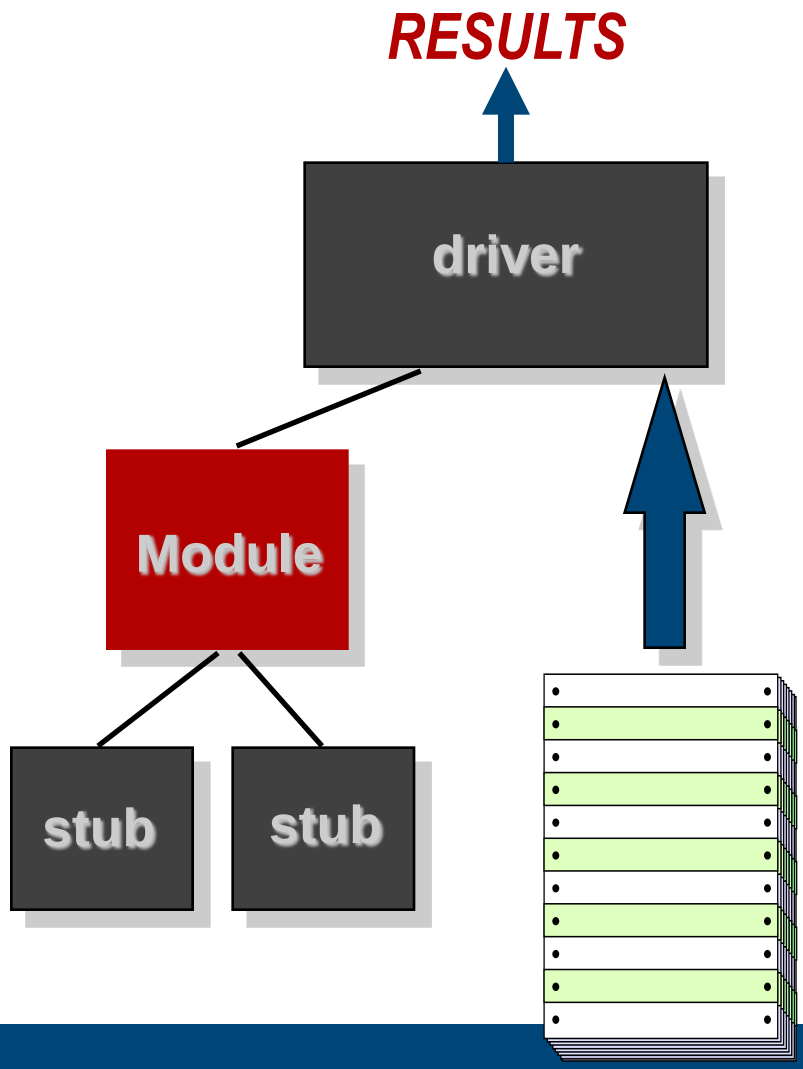
test cases

Unit Test Environment

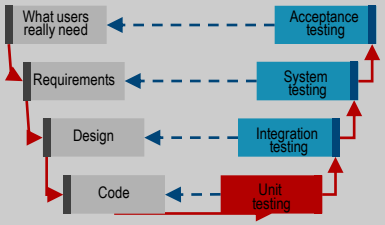


JACOBS UNIVERSITY

- 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

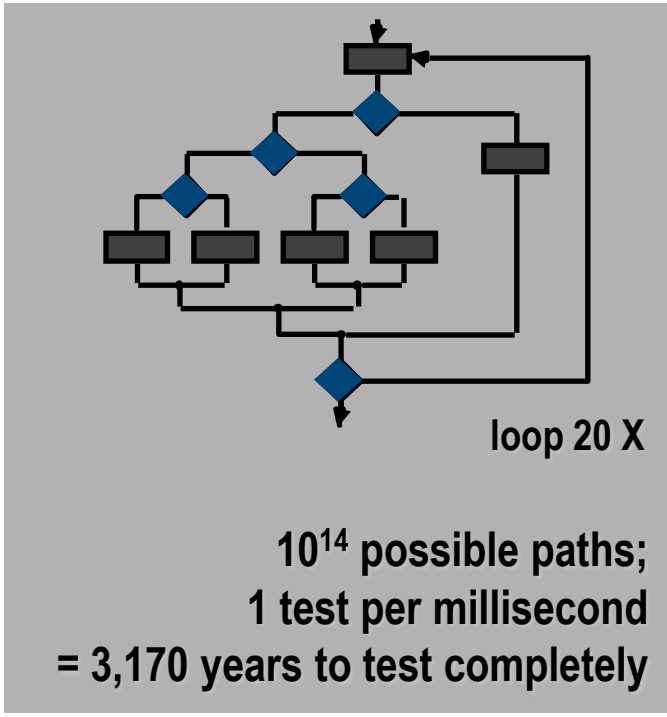


Equivalence Class Testing



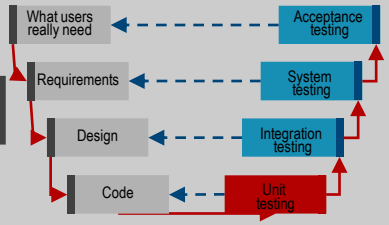
JACOBS UNIVERSITY

- Practically never can do exhaustive testing on input combinations



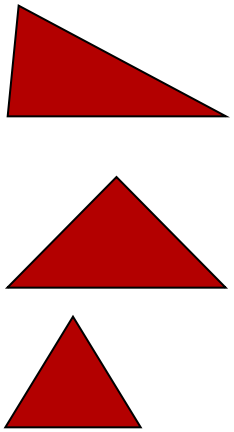
- How to find „good“ test cases?
  - Good = likely to produce an error
- Idea:
  build equivalence classes
  of test input situations,
  test one candidate per class
  - See lab

Test Your Testing, Reloaded



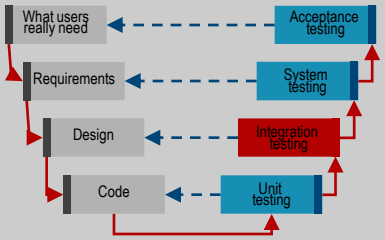
JACOBS UNIVERSITY

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



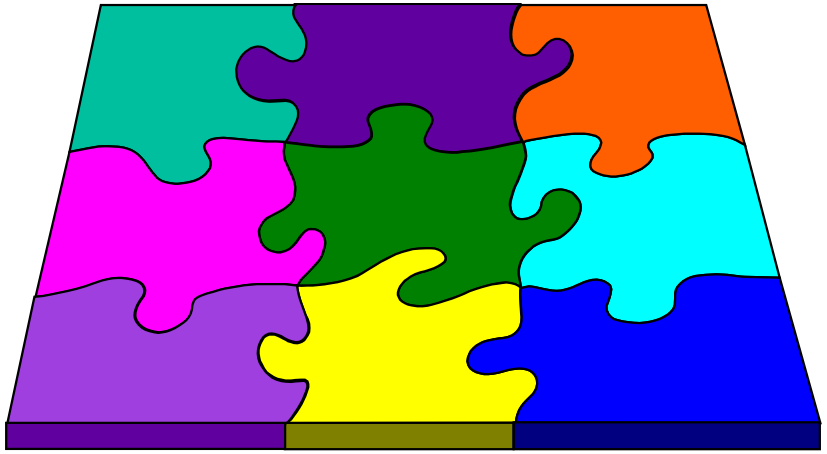
- ...test cases?

Integration Testing



JACOBS UNIVERSITY

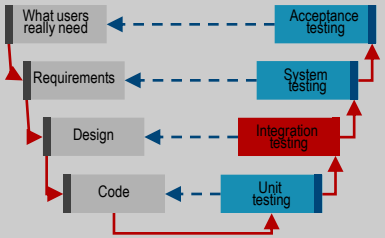
- Integration testing
  = test interactions among units
  - Import/export type compatibility
  - range errors
  - representation
  - *…and many more*



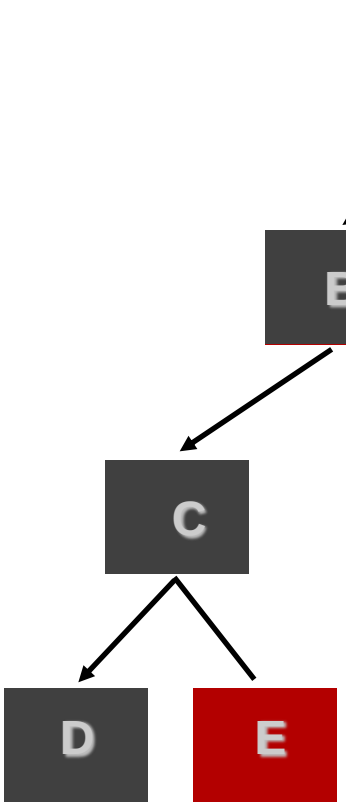
- Sample integration problems
  - F1 calls F2(char[] s) -- F1 assumes array of size 10, F2 assumes size 8
  - F1 calls F2(elapsed_time) -- F1 thinks in seconds, F2 thinks in miliseconds

- Strategies: Big-bang, incremental (top-down, bottom-up, sandwich)

Top-Down Integration



JACOBS UNIVERSITY

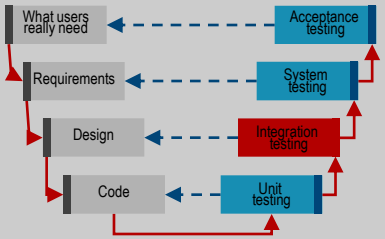


top module is tested with stubs

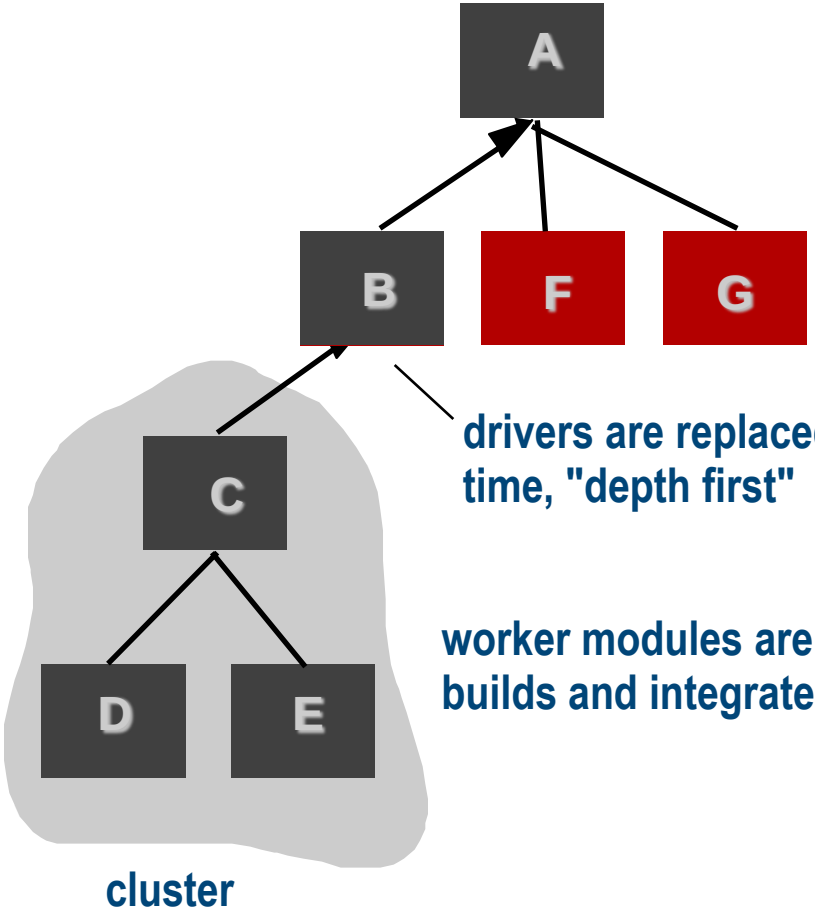
stubs are replaced one at a time, "depth first"

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

Bottom-Up Integration



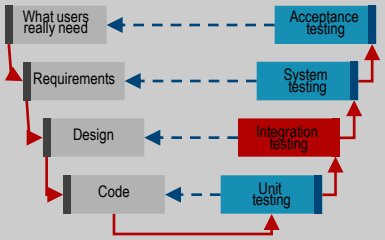
JACOBS UNIVERSITY



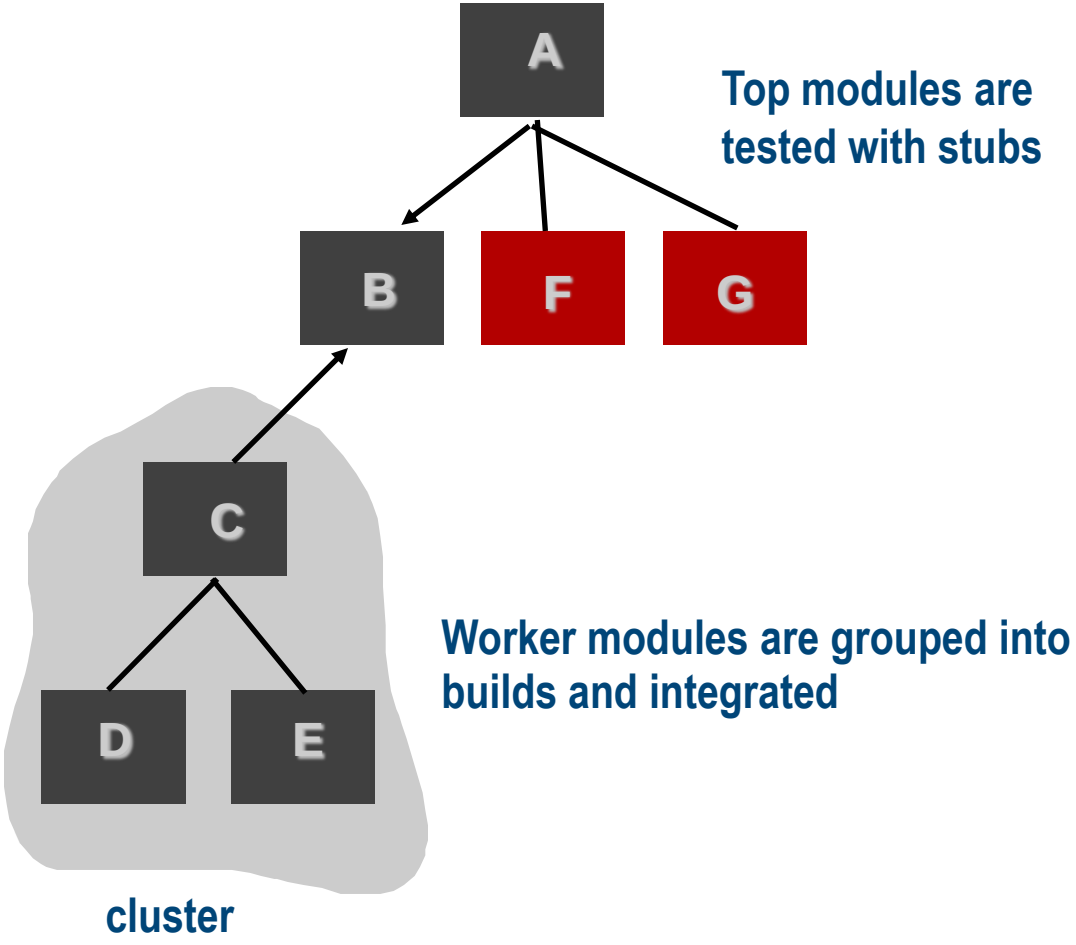
drivers are replaced one at a time, "depth first"

worker modules are grouped into builds and integrated

Sandwich Testing



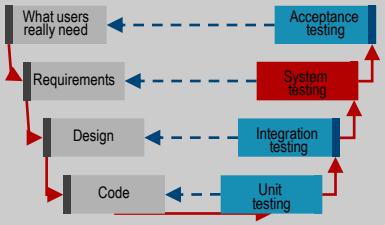
JACOBS UNIVERSITY



Top modules are tested with stubs

Worker modules are grouped into builds and integrated

System Testing

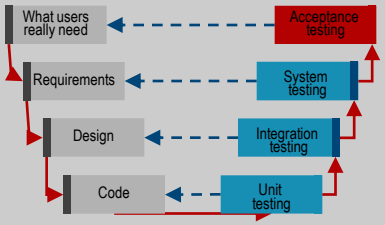


JACOBS UNIVERSITY

- 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



JACOBS UNIVERSITY

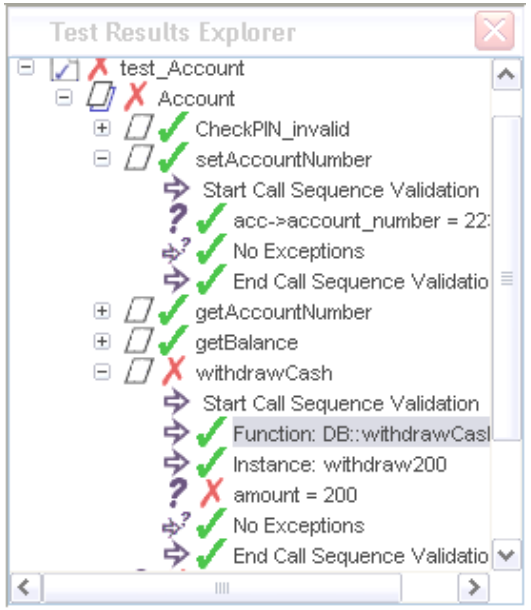
- Goal: Get approval from customer
  - try to structure it!
- be suresuresure that the demo works
- Customer may be tempted to demand more functionality when getting exposed to new system
  - Ideally: get test cases agreed already during analysis phase
  - …will not work in practice, customer will feel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

Testing Methods

static
dynamic
regression

JACOBS UNIVERSITY

- 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

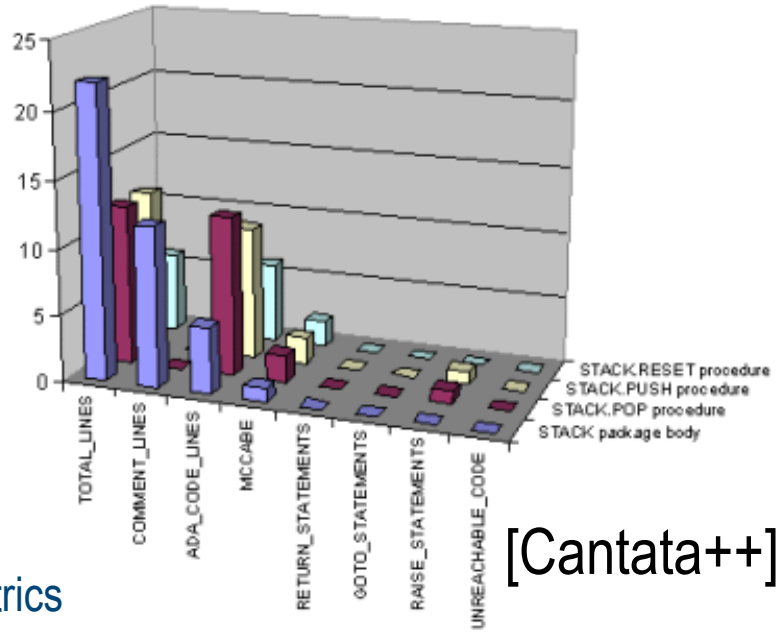


Function: `bool enoughCash(int )` **FAIL**
Location: `W:\cgi-bin\src\unit_account\account.cpp`
Scope: `Account`

	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%

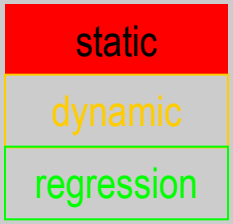
Static Analysis

- 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



JACOBS UNIVERSITY

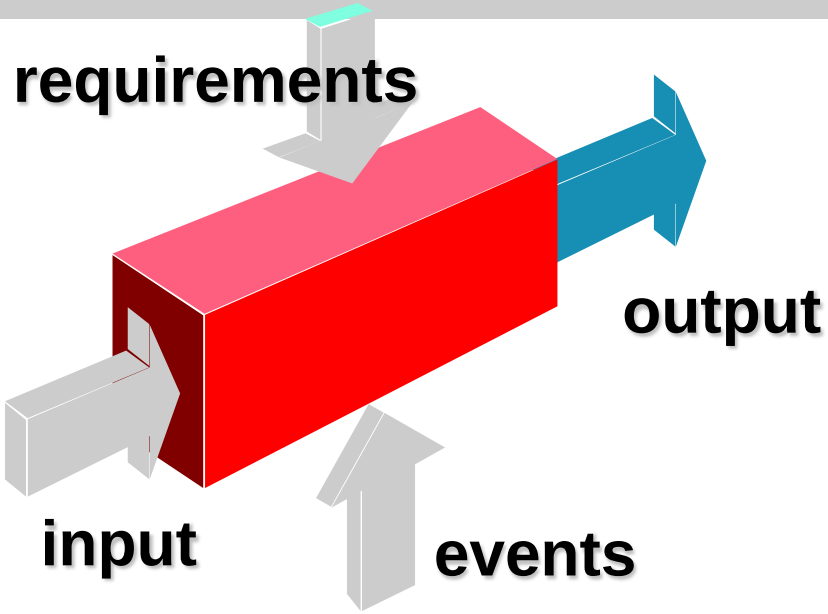
- Given a model of a program and a property, determine whether model satisfies property, based on mathematics
  - algebra, logic, …
  - *See earlier (invariants) and later!*
- Examples
  - Safety
    - *If the light for east-west is green, then the light for south-north should be red*
  - Liveness
 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

Black-Box = Spec-Based Testing

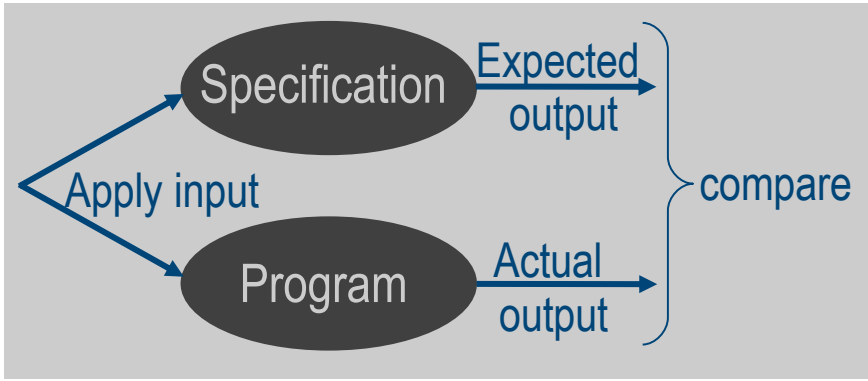
static
dynamic
regression

JACOBS UNIVERSITY

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



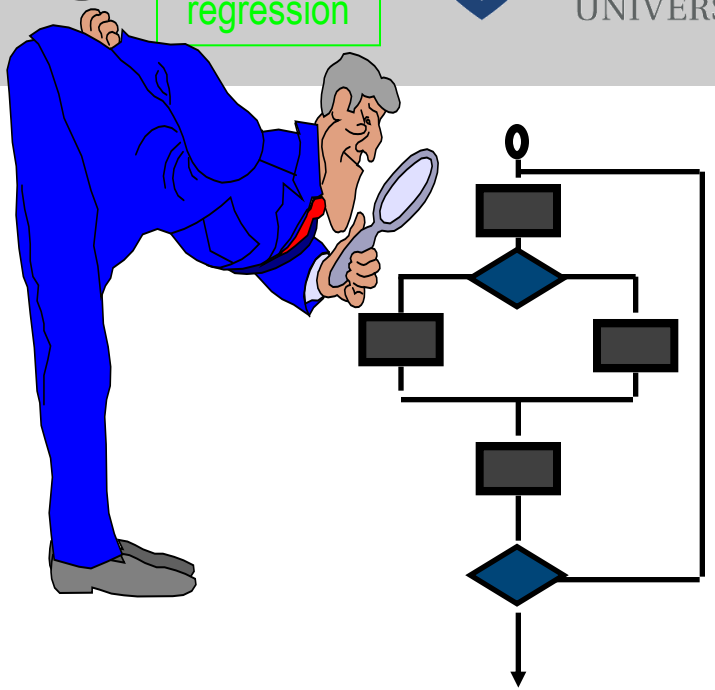
- Limitations
  - Specifications are not usually available
  - Many companies still have only code, there is no other document



White-Box (Glass-Box) Testing

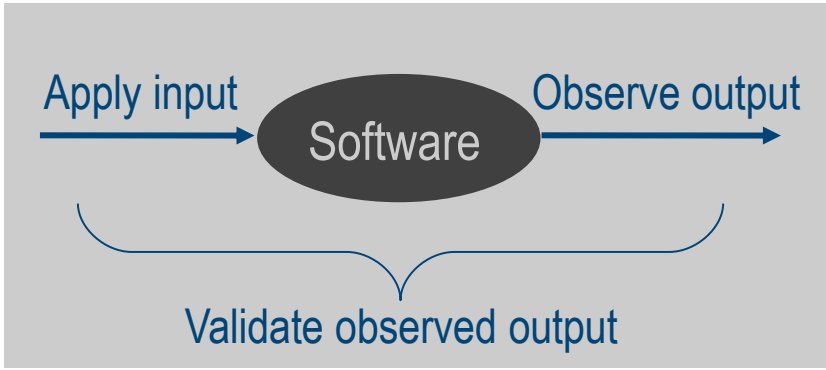
static
dynamic
regression

JACOBS UNIVERSITY



- 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?



Coverage Analysis

static
dynamic
regression

JACOBS UNIVERSITY

- 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! ↳)

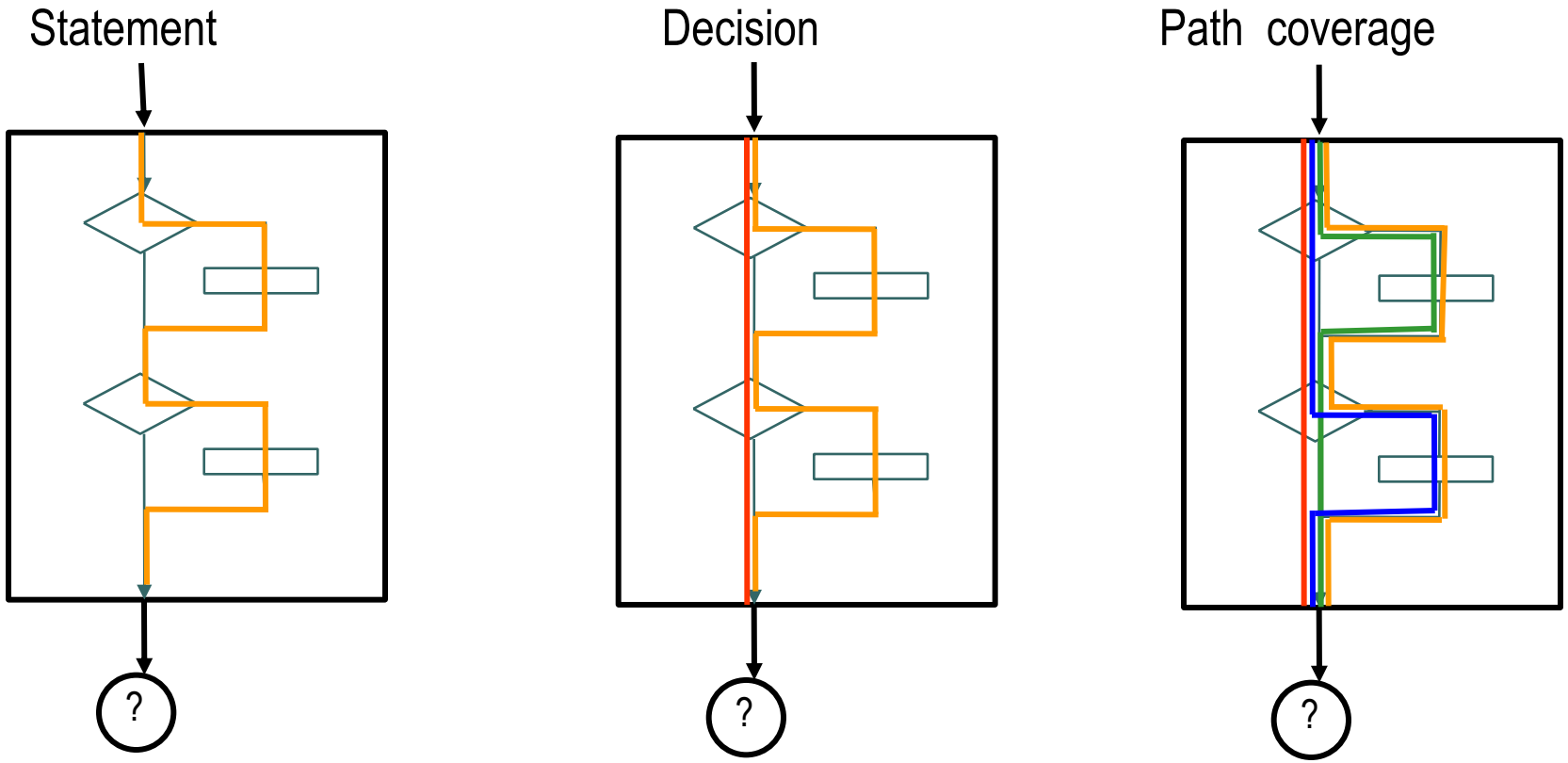
Source File - W:\cgi-bin\src\unit_account\account.cpp

```
22 /*
23  * Connect to the database and
24  * Check pin is correct
25  */
26     if (db->connect(DB_HOST,
27                     DB_USER,
28                     DB_PASS)) {
29         pinValid = db->checkPin(pin,
30                                 getAccountNumber());
31     }
32
33     return pinValid;
34 }
35
36 double Account::getBalance() const {
37     return db->getBalance(getAccountNumber());
38 }
39
40 //
41 // Check that there is enough cash (greater than or equal to
42 // requested amount)
43 //
44 bool Account::enoughCash(int cash) {
```

Coverage Analysis: Metrics

static
dynamic
regression

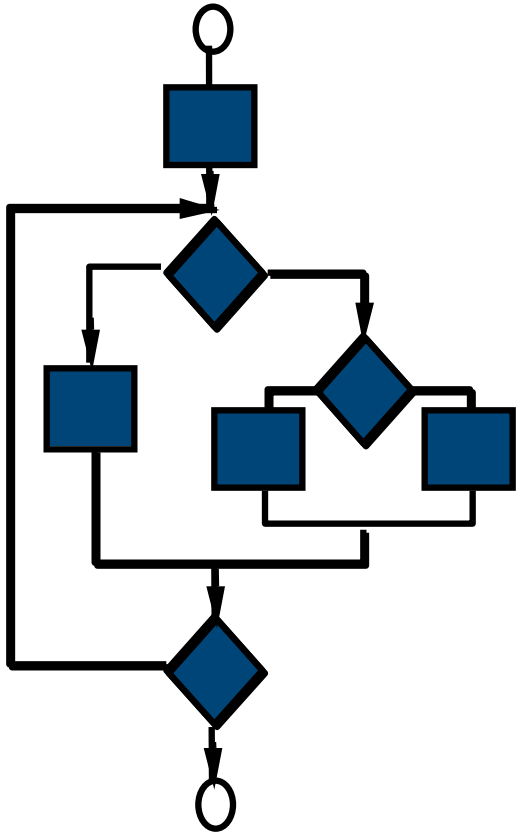
JACOBS UNIVERSITY



test cases?

Path Testing

- cyclomatic complexity of flow graph:
- V(G) = number of simple decisions + 1
  - V(G) = number of enclosed areas + 1



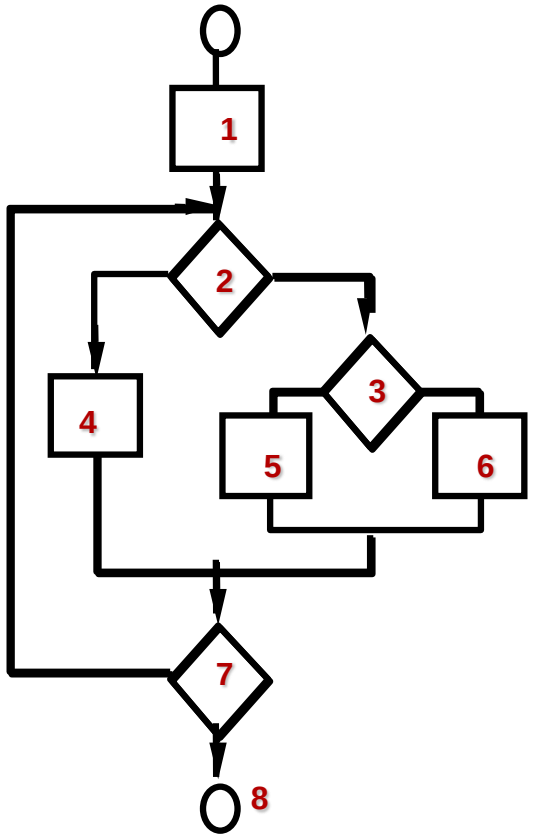
- In this case, V(G) = ?

Path Testing

static
dynamic
regression

JACOBS UNIVERSITY

- 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

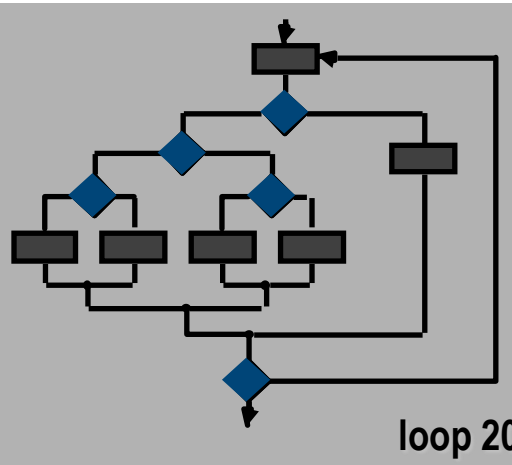


Equivalence Classes

static
dynamic
regression

JACOBS UNIVERSITY

- Practically never can do exhaustive testing on input combinations



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

- 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
  - See lab

Terminology: Cx

What would you test?

JACOBS UNIVERSITY

- C0 = every instruction
- C1 = every branch (even if there's no `else`!)
- C2, C3 ~= every condition once `true`, once `false`
  - **Numbering historically grown, C1 & C2 not related!**
- C4 = path coverage: every possible path taken (↳ if/if example)

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

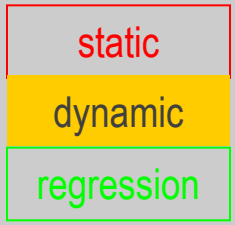
Example: DO-178B

JACOBS UNIVERSITY

- 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	

Tech Inset: Memory Leaks



JACOBS UNIVERSITY

- Memory leak = memory gets allocated, but not released any more
- Why bad?
  - Reduces performance due to excessive resource usage
  - May cause crashes (memory overflow, quota)
- Side note: Pointer errors form one of the biggest problem sources in C/C++ and similar languages
  - Java doesn't have that!
- How to solve
  - Find out where exactly memory is leaked = why not released

PLs Revisited: Where are my Data?

static
dynamic
regression

JACOBS UNIVERSITY

- Stack
  - automatic management (stack frame allocation/deallocation)
  - stack pointer marks limit of valid data on stack;
    compiler generates code to grow & shrink stack
    on function entry/return (generally: block level)
  - *local variables, function return addresses*
- Heap
  - explicit allocation and deallocation (programmer driven)
    using malloc() / free or new / delete / delete[]
  - *pointer targets*

Memory Leak: Example

static
dynamic
regression

JACOBS UNIVERSITY

- Variable created by “usual declaration” sits on stack

```
void f()
{
    int i = 3;          // memory for i and obj
    MyObject obj;       // allocated on the stack
  ...
}
```

  - Allocated by increasing stack ptr = reserving memory
  - Deallocated by decreasing stack ptr = releasing memory

Memory Leak: Example

static
dynamic
regression

JACOBS UNIVERSITY

- Dynamic allocation with memory leak

```
void f()
{
    int i = 3;                      // memory for i and obj
    MyObject obj;                   // allocated on the stack

    MyClass *ptr;                   // ptr on stack
    ptr = new MyClass( args );      // creates object on heap, writes address into ptr

    …

    return;                         // ptr vanishes, object remains – mem leak!
}
```

Memory Leak: Example

static
dynamic
regression

JACOBS UNIVERSITY

- Dynamic allocation without memory leak

```
void f()
{
    int i = 3;                      // memory for i and obj
    MyObject obj;                   // allocated on the stack

    MyClass *ptr;                   // ptr on stack
    ptr = new MyClass( args );      // creates object on heap, writes address into ptr

    …

    delete ptr;                     // heap memory is freed
    ptr = NULL;                     // because we are orderly

    return;                         // ptr vanishes, no object remains
}
```

Memory Leak: Tool Support

static
dynamic
regression

JACOBS UNIVERSITY

- Instrument object code, find mem leaks
  - At some runtime expense
- Valgrind, Purify etc.
- Purify: 12 illegal memory access types
  - Read or write without allocation
  - Read or write after deallocation
  - Read without previous write
  - …

…*and then*: PL Particularities

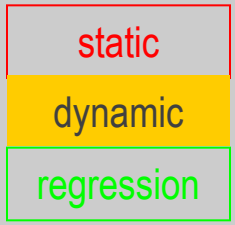
static
dynamic
regression

JACOBS UNIVERSITY

```
public static void main(String[] args){
      int imax=Integer.MAX_VALUE;
      int imax1=imax+1;
      double dmax=Double.MAX_VALUE;
      double dmax1=dmax*100.;
      double dmin1=-dmax1;
      double aha1=dmax1/dmax1;
      double aha2= 3./0.;
      System.out.println("imax: "+imax);
      System.out.println("imax1: "+imax1);
      System.out.println("dmax: "+dmax);
      System.out.println("dmax1: "+dmax1);
      System.out.println("dmin1: "+dmin1);
      System.out.println("aha1: "+aha1);
      System.out.println("aha2: "+aha2);
}
```

```
imax: 2147483647
imax1: - 2147483648
dmax: 1.7976931348623157E308
dmax1: Infinity
dmin1: -Infinity
aha1: NaN
aha2: Infinity
```

Performance Profiler



JACOBS UNIVERSITY

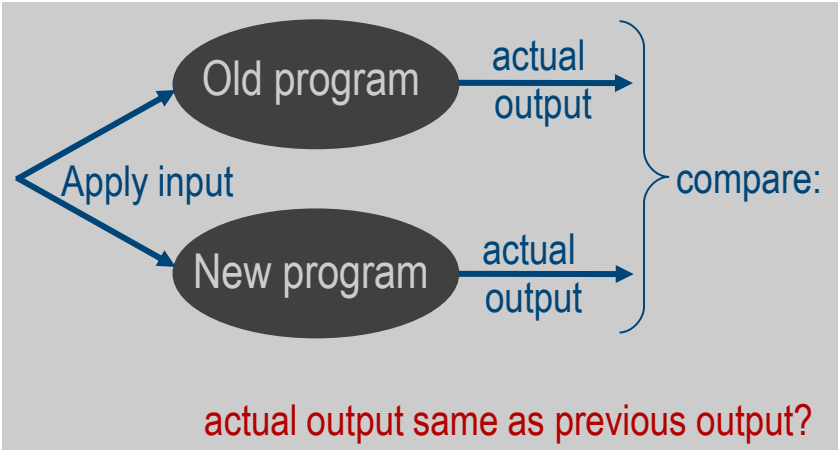
- Code profiling =
  benchmarking execution to understand where time is being spent
- Questions answered by profiling:
  - Which lines of code are responsible for the bulk of execution time?
  - How many times is this looping construct executed?
  - Which approach to coding a block of logic is more efficient?
- Without profiling, answering this becomes a guessing game
- Technique:
  - Profiler runs while application runs
  - records frequency & time spent in each line of code
  - Generates log file

Regression Testing

static
dynamic
regression

JACOBS UNIVERSITY

- 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
  - Ex: store previous log output, do 'diff'
  - Advantage: easy automatic testing
- Limitations
  - Finds only deviations, cannot judge on error
  - Can only find newly introduced deviations
  - Only reasonable for fully automated tests



Test Organization

JACOBS UNIVERSITY

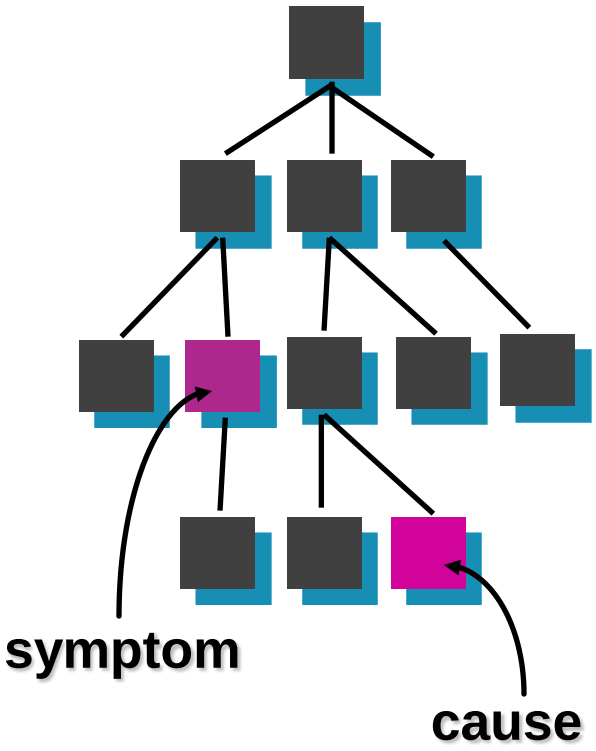
- Tests should be self-sustaining
  - create your own data,
  - ...and clean up
  - Expect nothing!
- Set up controlled enviroment
  - data sets, files, environment variables, system configuration, ...
  - excellent for repeatability of complex setup: virtual machines (eg,VMware box)

Create Testable Software!

JACOBS UNIVERSITY

- 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

Symptoms & Causes (=Nightmares)

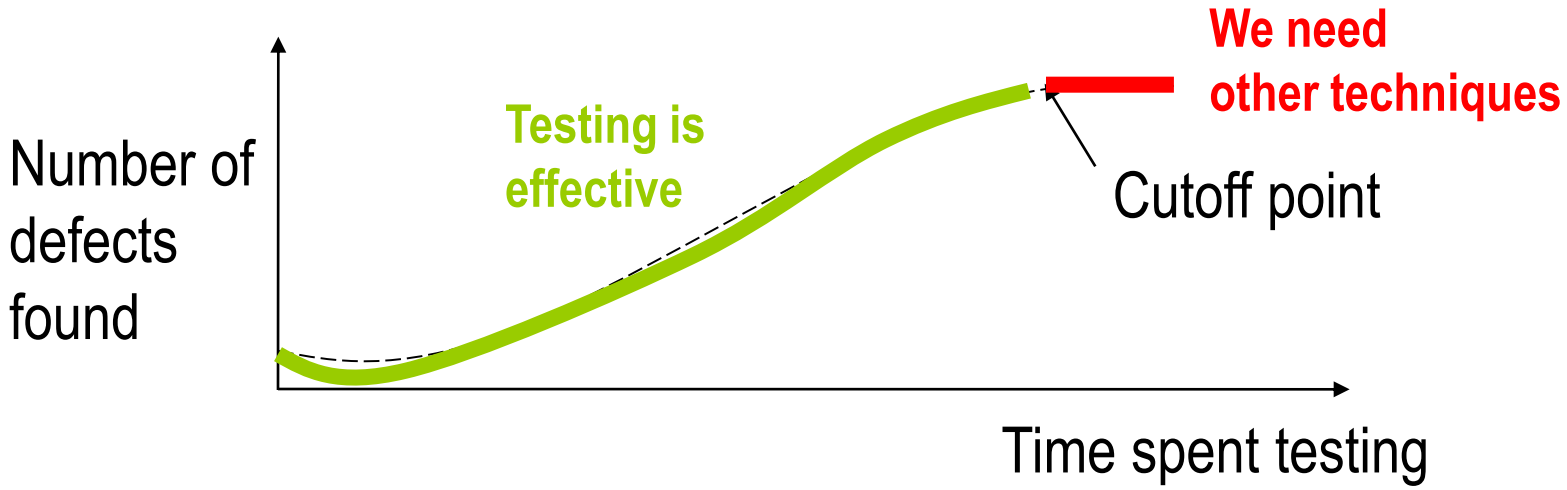


- symptom and cause may be geographically separated
- symptom may disappear when another problem is fixed
- symptom may be intermittent
- cause may be due to a combination of non-errors
- cause may be due to a system or compiler error
- cause may be due to assumptions that everyone believes

Economics of Testing (I)

JACOBS UNIVERSITY

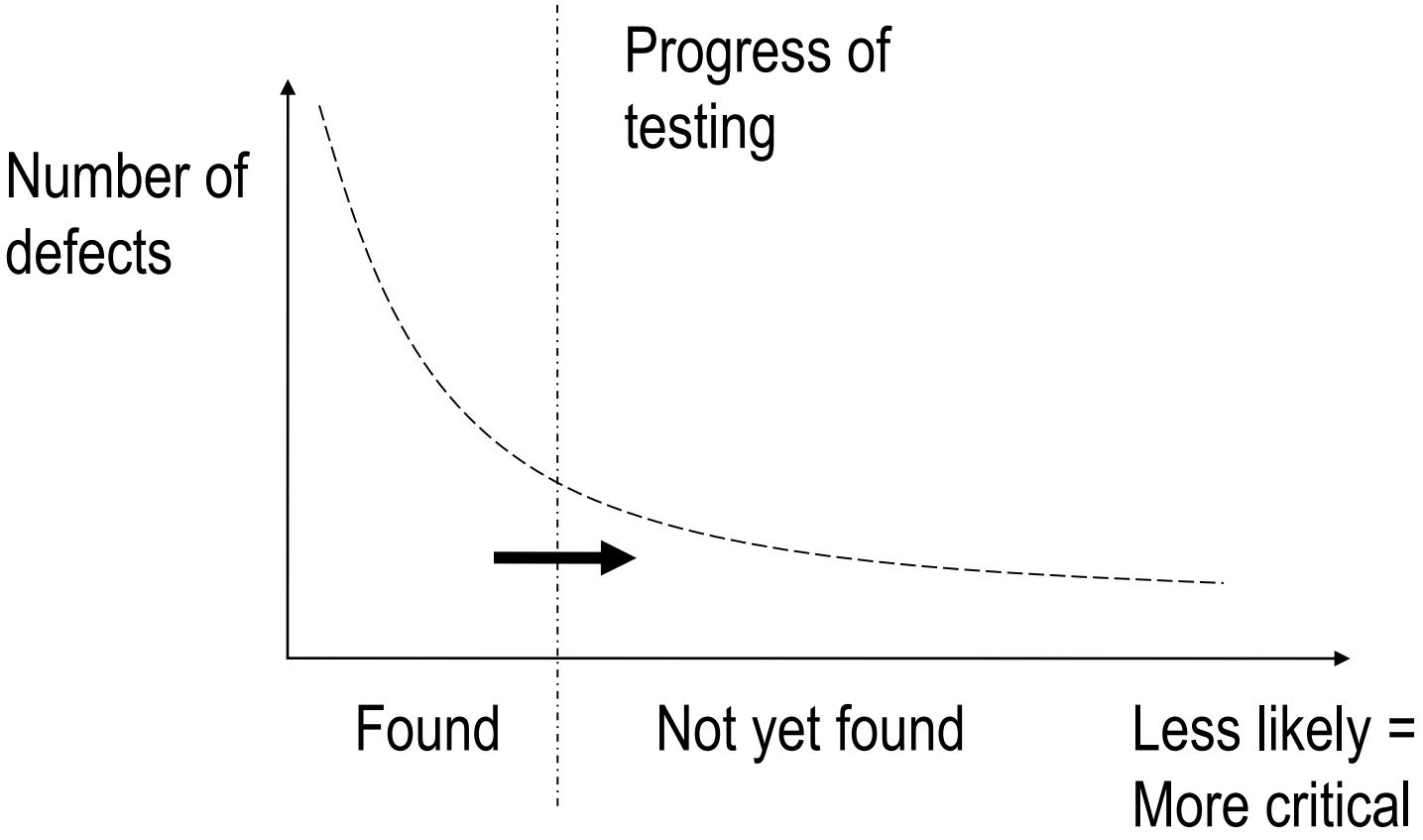
- The characteristic S-curve for error removal



Economics of Testing (II)

JACOBS UNIVERSITY

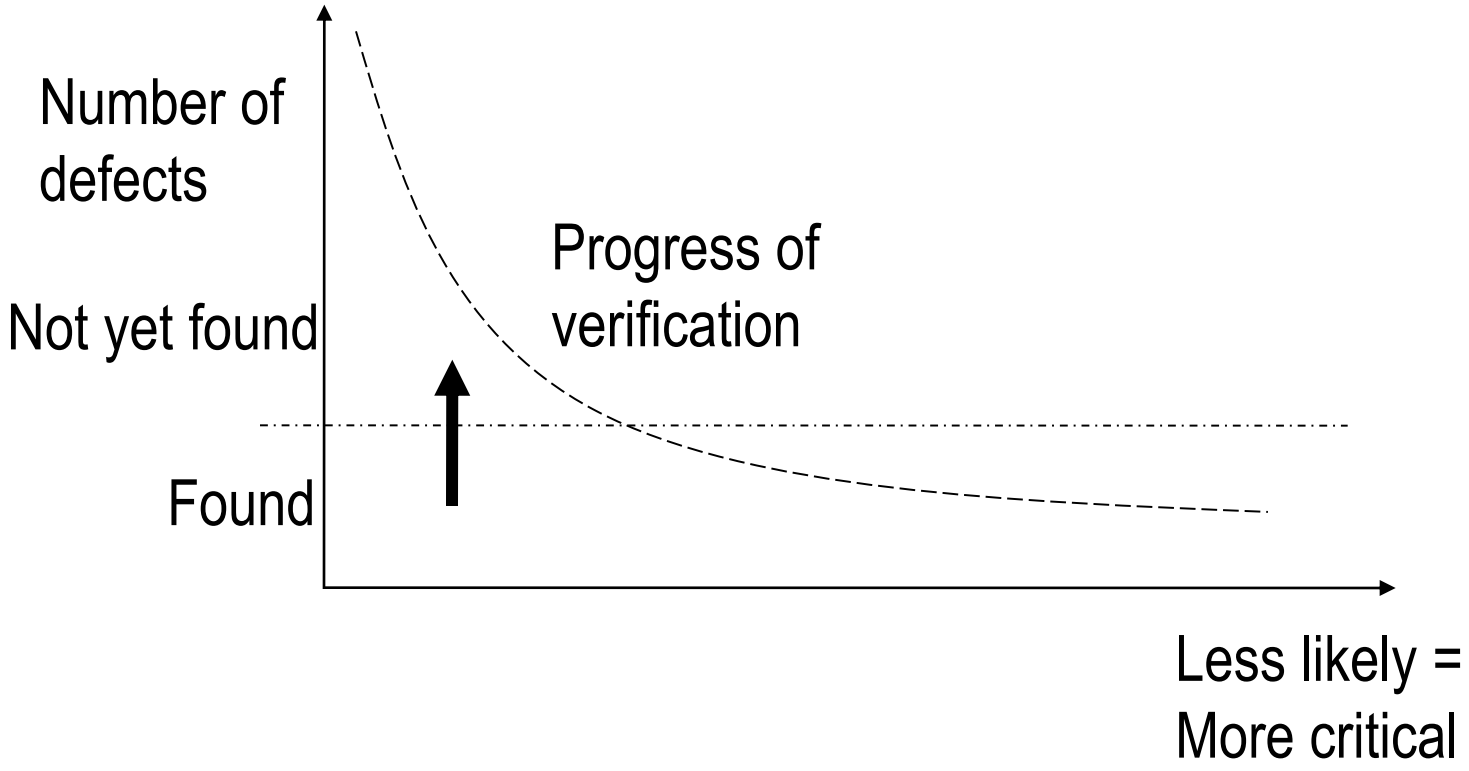
- Testing tends to intercept errors in order of their probability of occurrence



Economics of Testing (III)

JACOBS UNIVERSITY

- Verification is insensitive to the probability of occurrence of errors



Summary

- 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

Summary (contd.)

JACOBS UNIVERSITY

- Final Thoughts [Pressman]
  - Think about what you see
  - Use tools to gain more insight
  - If at an impasse, get help from someone else
  - Be absolutely sure to conduct 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
- “*Testing is successful if the program fails*” - Goodenough & Gerhart, 1975
- "*Testers are customer advocates*“ – n.n.