Software Testing

Changing Your Testing as Your World Changes

A diagnostic quiz

- How much of this set of topics do you already know?

- We'll repeat at the end of the session
- (and I'll show you evidence of the value of ending with a quiz)

- Answer the quiz
- Work alone

5 minutes (ish)

Diagnostic Quiz

1. What are three kinds of software systems that each need very different testing?
2. What is “exploratory testing”?
3. What is “specification-based testing” aka “requirements-based testing”?
4. When you are fuzz testing (pouring random inputs into a program interface), you often can’t know in advance what the correct response to an input it is. How can it still be useful in that case?
5. What effect does a pure waterfall development lifecycle have on testing?
6. What’s a good reason to stop testing?

So, you've written down your answers...

- Keep them

- I'll now give you mine
- (but in the form of a set of lectures)

Section 1 – Software Development Happens in Many Different Worlds

We've been talking about testing software

- All software?

- “Five Worlds” by Joel Spolsky
  - http://www.joelonsoftware.com/articles/FiveWorlds.html
  - Shrinkwrap (open source; configurable; web-based)
  - Internal
  - Embedded
  - Games
  - Throwaway

Q – What does this mean for testing practice?

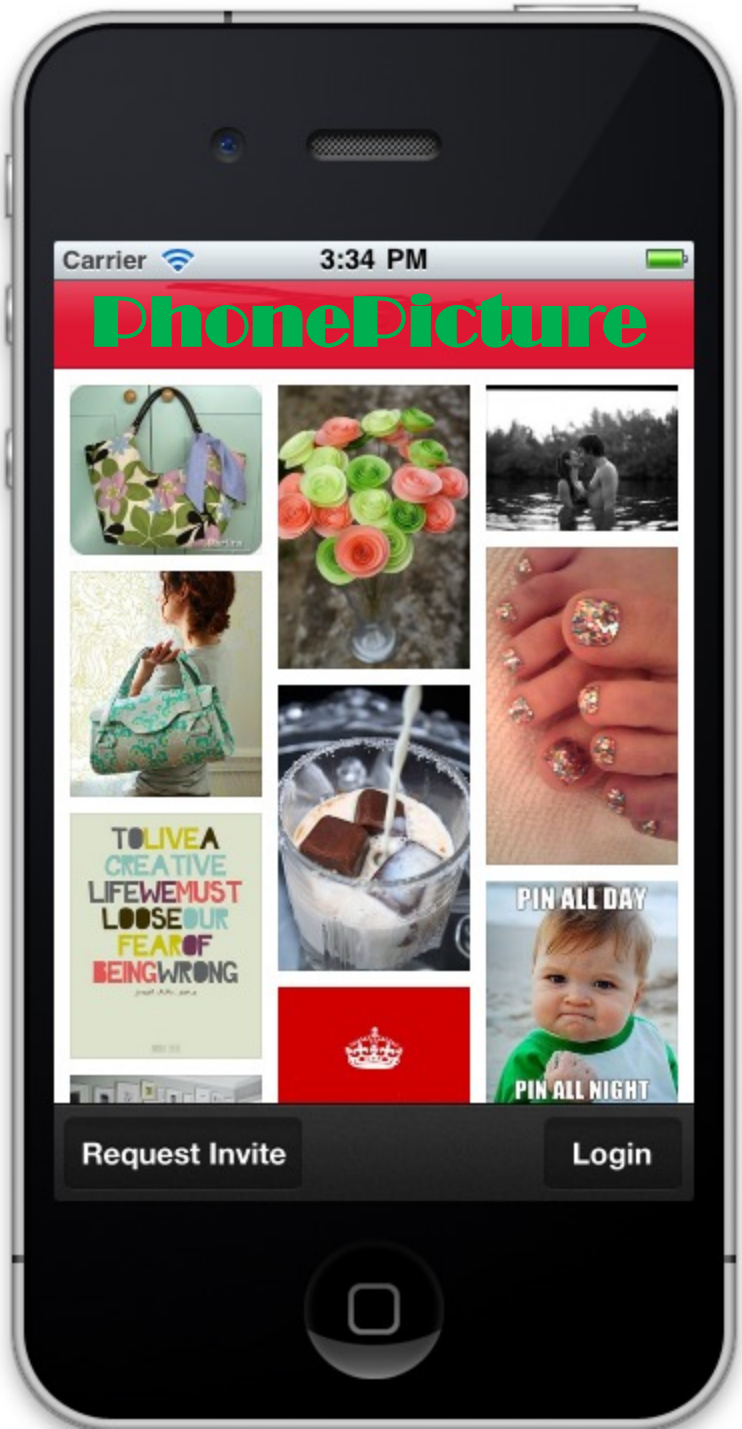
- You need to think about how you test – every time
- You need to vary how you test based on the project and context

Introducing…

PhonePicture*

- Manage pictures on your phone!
- Send them to your friends!
- Rate photos online!
- Win prizes!
- Your private photos are encrypted!

(*definitely different from Instagram etc)


Carrier
3:34 PM
PhonePicture
TOLIVEA
CREATIVE
LIFEWEMUST
LOOSEOUR
FEAROF
BEINGWRONG
PIN ALL DAY
PIN ALL NIGHT
Request Invite
Login

Brainstorm for 2 minutes

- What kind of software is this?
  - Shrinkwrap
  - Internal
  - Embedded
  - Games
  - Throwaway
- Why?
- What are some implications of that, for testing?

Section 2 – There are a Great Many Techniques for Testing

We have a program with faults in it

- How should we test it?

Definition

Exploratory Testing

Exploratory testing is simultaneous learning, test design, and test execution.

Exploratory testing…

I once had the mission of testing a popular photo editing program in four hours. My mission was to assess it against the specific standards of the Microsoft Windows Compatibility Certification Program. The procedure for performing such a test is laid out as a formalized exploratory testing process. My goal was to find any violations of the compatibility requirements, all of which were clearly documented for me.

With this rather clear charter in mind, I set myself to test. Applying one of the simplest heuristics of exploring, I chose to begin by walking through the menus of the application, trying each one. While doing so, I began creating an outline of the primary functions of the product. This would become the basis for reporting what I did and did not test, later on.

I noticed that the Save As… function led to a very sophisticated set of controls that allow the user to set various attributes of image quality. Since I knew nothing about the technical aspects of image quality, I felt unprepared to test those functions. Instead, I started an issues list and made a note to ask if my client was willing to extend the time allowed for testing so that I could study documentation about image quality and form a strategy for testing it. Having made my note, I proceeded walking through menus.

http://www.satisfice.com/articles/et-article.pdf

… in …

A basic strategy of ET is to have a general plan of attack, but allow yourself to deviate from it for short periods of time. Cem Kaner calls this the "tour bus" principle. Even people on a tour bus get to step off it occasionally and wander around. The key is not to miss the tour entirely, nor to fall asleep on the bus. My first urge to leave the tour of the menus happened when I found a dialog box in the program that allowed me to control the amount of memory used by the application. This immediately gave me an idea (sudden ideas are encouraged in exploratory testing). Since one of the requirements of the Windows Compatibility program is stability, I thought it would be useful to set the product to use the minimum amount of memory, then ask it to perform memory intensive functions. So I set the slider bar to use 5% of system memory, then visited the image properties settings and set the image size to 100 inches square. That's a big canvas. I then filled the canvas with purple dots and went over to the effects menu to try activating some of the special graphical effects.

Okay, here comes an important part: I chose a "ripple" effect from the menu and *bam*, the product immediately displayed an error message informing me that there was not enough memory for that operation. This is very interesting behavior, because it establishes a standard.

http://www.satisfice.com/articles/et-article.pdf

Action

I have a new expectation from this point forward: a function should be able to prevent itself from executing if there is not enough memory to perform the operation. This is a perfect example of how, in exploratory testing, the result of one test influences the next, because I then proceeded to try other effects to see if the rest of them behaved in the same way. Verdict? None of the others I tried behaved that way. Instead, they would crank away for five minutes, doing nothing I could see other than drive the hard disk into fits. Eventually an error popped up "Error -32: Sorry this Error is Fatal." and the application crashed.

This is a nice result, but I felt that the test wouldn't be complete (exploratory testers strive to anticipate questions that their clients will ask later on) unless I set that memory usage lever all the way up, to use the most memory possible. To my surprise, instead of getting the Error -32, the entire operating system froze. Windows 2000 is not supposed to do that. This was a far more serious problem than a mere crash.

At this point in the process, I had spent about 30 minutes of a 4 hour process, and already found a problem that disqualified the application from compatibility certification. That was the good news. The bad news is that I lost my test notes when the system froze. After rebooting, I decided I had learned enough from stress testing and returned to the menu tour.

http://www.satisfice.com/articles/et-article.pdf

Exploratory testing is a skilled practice

- … and has no concept of “completeness”
- A strength is its ability to “follow a script” yet diverge.
- You can use it to discover design/requirements, and to answer questions the customer may not know they want answered.

- But…

- You might want more systematic techniques
- You might want to supplement tester skill with explicit processes

Definition

Specification-based Testing

Testing software against all the explicit claims made about it in the specification

Why is spec-based testing good?

- Systematic
- Specification is usually simpler than the software itself (well, if it's a good spec)
- Specification *hopefully* captures the things that matter most about the software

Informal Specification

Brief Summary:

- Rent A Coder reminder: You MAY NOT post the final solution for this (and any) project before your bid is accepted and funds are fully escrowed. Anyone who does may have their account permanently suspended. However, you CAN post:
  - On programming projects: A prototype or functional demo...as long as source code is not provided.
  - On graphics projects: A watermarked and low-resolution version of the work.

I have many customers that run several applications I have written. Each app now makes log entries to a SQL table when they start, stop & are running. I would like a page written in classic asp that looks through the SQL table and reports back the status of each customer and each application. This should show me at-a-glance who is fine and who has problems.

Most of the design will be left up to the coder, I would just like it to be easy to understand and show me very quickly where the problems are.

The SQL log file reporting is a new feature so I will work with the coder to make any changes based on ther suggestions for better reporting. This will run on a server running IIS and the tabe is in MS SQL. I have attached the sql table shema, sample data & a screen shot of some of the data.

The asp page should run in both IE7 and IE8. No other browser compatibilty is needed.

Contracts as Specifications: Eiffel

The put method for a class DICTIONARY[ELEMENT] in Eiffel:

```
put (x: ELEMENT; key: STRING) is
                -- Insert x so that it will be retrievable through key.
        require
                count <= capacity
                not key.empty
        do
                ... Some insertion algorithm ...
        ensure
                has (x)
                item (key) = x
                count = old count + 1
        end
```

http://www.eiffel.com/developers/design_by_contract_in_detail.html

Contracts as specifications: SPARK

```
procedure Increment (X : in out Counter_Type);
--# derives X from X;
--# pre X < Counter_Type'Last;
--# post X = X~ + 1;
```

http://en.wikipedia.org/wiki/SPARK_(programming_language)

Not all specs are great for testing from

- E.g. there may be no *mechanical* way to convert an informal spec to tests.
- E.g., “the switch must deliver reasonable performance under typical loads”
  - “Fit criteria” later.
- Code with specs embedded in it (like Eiffel, Spark, JML etc) has a number of advantages both for debugging and testing.
  - Code+spec don’t get out of sync, for one.

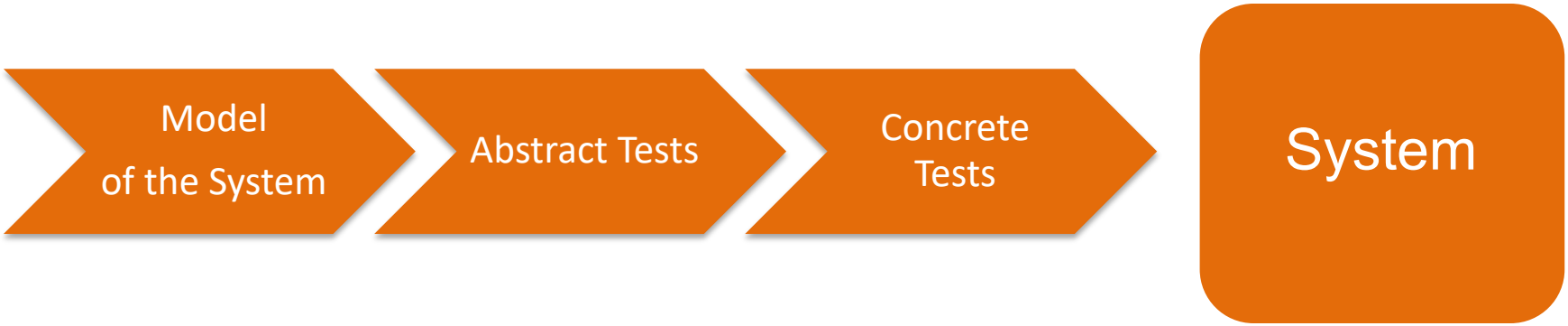
Definition

Model-based Testing

Using models of expected behaviour to produce test-case specifications that can reveal discrepancies between actual program behaviour and the model.

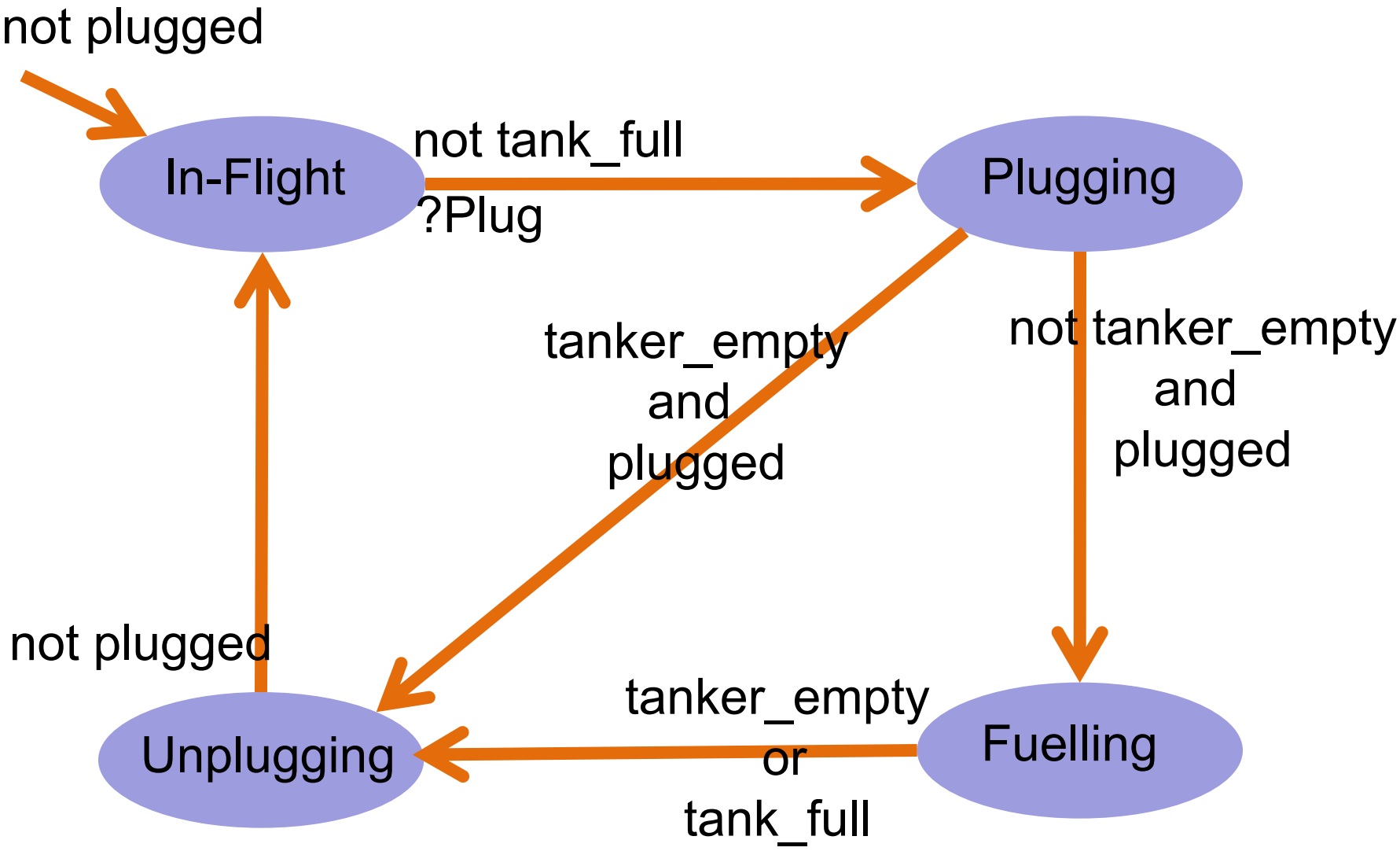
Model-Based Testing

- Goal: Test a program specified by a model to find faults
- Principle:
  - The software is supposed to be specified by a model.
  - The testing checks that the program conforms to the model.
  - Abstract test cases are derived from the model.
  - Concrete test cases are translated from the abstract ones.
  - Test cases are executed to check the conformance



An in-flight refuelling system

- A fighter plane can be refuelled while in flight
- The plane is flying, it plugs the in-flight fuel entry to the tanker plane.
- The fuel is poured into the fighter plane.
- When it is full or there is no more fuel in the tanker, the fighter plane unplugs its fuel entry.
- Plugging/unplugging takes less than 10min.


not plugged
In-Flight
not tank_full
?Plug
Plugging
tanker_empty
and
plugged
not tanker_empty
and
plugged
not plugged
Unplugging
tanker_empty
or
tank_full
Fuelling

Q – what if the spec or model is wrong or incomplete?

- Or if there wasn't one in the first place.

Definition

Fuzz Testing

Testing by feeding in random inputs until the program crashes or violates assertions.

Will it be effective on:

```
/**
 * Adds a value to draw.
 *
 * @param x the x value.
 * @param y the y value
 */
public void addValue(double x, double y) {
    // if the maximum x is too low we extend it.
    if (maxX<x) {
        maxX=1.2*x;
        nDigitsX=(int)Math.floor(Math.log10(maxX));
    }
    // if the maximum y is too low, we extend it.
    if (maxY<y) {
        maxY=2*y;
        nDigitsY=(int)Math.floor(Math.log10(maxY));
        leftBorder = 20+nDigitsY*7;
    }
    // we add the point to the graph
    series[0].add(x);
    series[1].add(y);
}
```

And here?

```
/**
 * Static method creating an instance of a given Equation type.
 * The method asks values of parameters through option panes.
 *
 * @param equationType the class of the equation
 * @return the Equation
 */
@SuppressWarnings("unchecked")
public static Equation createEquationFromType(Class equationType) {
    Constructor c =  equationType.getConstructors()[0];
    int n_arguments = c.getParameterTypes().length;
    Object[] arguments = new Double [n_arguments];
    for (int i=0; i<n_arguments;i++) {
        //ask for values
        String s = JOptionPane.showInputDialog(null, ((char)(((byte)'a')+i))+" =",
                "Enter argument", JOptionPane.QUESTION_MESSAGE);
        arguments[i] = Double.parseDouble(s);
    }
    try {
        // we return the new instance
        return (Equation)c.newInstance(arguments);
    } catch (Exception e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(null, "alert", "alert", JOptionPane.ERROR_MESSAGE);
    }
    // if there was no exception
    return null;
}
```

Fuzz Testing and Test Oracles

- Test Oracle – something that tells you whether a test result is good or not
  - Can be known correct result ("add(2,2) should return 4")
  - Can be existing reference implementation
  - Can be simple but slower implementation used to test optimized implementation
- Fuzz testing doesn't normally use oracles as such
  - Though can use an oracle that checks validity of results ("could this result *possibly* be right?")
  - See http://www.squarefree.com/2014/02/03/fuzzers-love-assertions/ for some discussion of using assertions for this in the Firefox codebase

Class Exercise – 3-5 minutes

- *Recall PhonePicture*
- Sketch how you could apply each of the following testing types to PhonePicture:
  - Exploratory testing
  - Specification-based testing
  - Model-based testing
  - Fuzz testing

PhonePicture – *your pictures, your phone* (also on our servers)

Fuzz testing (often) ignores our knowledge about how the program works

Definition

Partition Testing

Testing by partitioning the input domain (based on their significance for the program) and selecting input values from those partitions.

Some Partitions?

- Int
- Boolean
- Character
- Float
- Double
- String
- (int, int)

Do they work for this?

```
int increment(int i){
        return i+1;
}
```

Partition Testing

- (Sometimes called equivalence partition testing.)
- Choosing partitions can be done either with or without knowledge of code.
- Partitions should ideally be disjoint but sometimes this is hard.

Partition Testing Example

```
int get_index_of(String s, char c);
// return first index of c in s
// otherwise return -1
```

- What partitions?
- What values for each partition?

Definition

Boundary Testing

Testing by choosing inputs that are on semantically significant boundaries.

Boundary Values for these?

- Int
- Boolean
- Character
- Float
- Double
- String
- (int, int)

Do they make a difference for this?

```
int increment(int i){
        return i+1;
}
```

How do we know if we're testing all of the software?

Definition

Coverage-based Testing

Testing until some measure of coverage reaches a pre-agreed target.

Coverage Targets

- "At least one test that involves…"
  - Every module/package
  - Every class
  - Every method
  - Every statement
  - Every exit (true/false) from every decision
    - If, while, case …
  - Every path through the program

Statement Coverage Examples

```
if(A) then
 F1();
F2();
```

Test Case: A=True
Statement Coverage achieved

```
int* ptr = NULL;
if (B)
   ptr = &variable;
*ptr = 10;
```

Test Case:
B=True
Statement Coverage Achieved

Problem : if B is false the code will fail with a null pointer exception.

Branch Coverage Examples

```
if(A)
  F1();
else
  F2();
if(B)
  F3()
else
  F4();
```

Test Cases for Branch Coverage:

A=T, B=T

A=F, B=F

```
if (A && (B || F1()))
  F2();
else
  F3();
```

Test Cases for Branch Coverage:

A=T, B=T

A=F

Problem: F1() never gets called. This problem occurs in languages with short circuiting boolean operators

Condition Coverage Examples

```
if(A && B)
  F1();
else
  F2();
if(C)
  F3()
else
  F4();
```

Test Cases for Condition Coverage:

A=T, B=T, C=F

A=F, B=F, C=T

```
if(A && B)
  F1();
else
  F2();
if(C)
  F3()
else
  F4();
```

Test Cases for Condition Coverage:

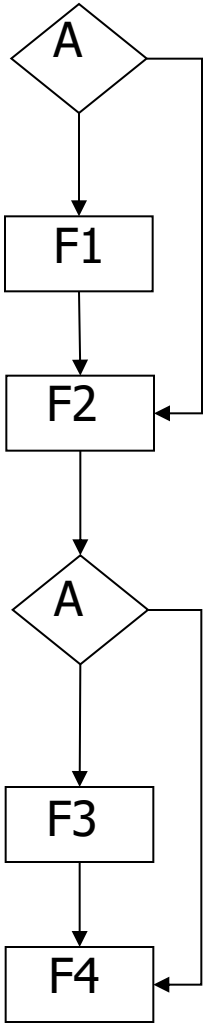
A=T, B=F, C=F

A=F, B=T, C=T

Problem: Branch coverage not achieved

Path Coverage Examples

```
if(A)
   F1();
F2();
if(A)
   F3();
F4();
```



Paths:

```
A-F1-F2-A-F3-F4
A-F2-A-F3-F4
A-F1-F2-A-F4
A-F2-A-F4
```

Problem: only two paths are feasible

```
A=T
A=F
```

Space product assurance

Software product assurance

http://esamultimedia.esa.int/docs/industry/SME/2005-Training/2-SWE-course/ECSS-Q-80B-10October2003.pdf

6.2.3 Handling of critical software

6.2.3.1

The supplier shall define and apply measures to assure the reliability of critical software.

These measures can include:

- use of software design or methods that have performed successfully in a similar application;
- failure mode analysis of the software, with the insertion of appropriate features for failure isolation and handling (see ECSS-Q-80-03);
- defensive programming techniques, such as input verification and consistency checks;
- prohibiting the use of language commands and features that are unpredictable;
- use of formal design language for formal proof;
- 100 % code branch coverage at unit testing level;
- full inspection of source code;
- witnessed or independent testing;
- gathering and analysis of failure statistics;
- removing deactivated code or showing through a combination of analysis and testing that the means by which such code could be inadvertently executed are prevented, isolated, or eliminated.

What if coverage criteria shows we're just not hitting some parts of the code?

Definition

Fault Injection

Testing by deliberately introducing faults into the software.

How does SFI work?

- *Legal* permutations or faults are input at interfaces (external and/or internal).
- Outputs show whether the injected fault propagates through the software.
- Requires *instrumentation* (software code) to observe the propagation process.

Uses for Software Fault Injection (SFI)

- Finding defects in software
- Robustness Testing
- COTS Validation/Determining failure modes
- Safety Verification
- Security Assessment
- Software Testability Analysis

SFI Examples

- Operating System Validation
  - Ballista (CM) – Linux and VxWorks robustness
  - Windows
- Network Security
  - NCSA httpd server
- Safety
  - Advanced Automatic Train Control system
  - Magneto Stereoaxis System

SFI can be used with or without source code

Variations

- Compile-time (whitebox) injection: modify source code to inject faults.
  - E.g., change **`a+=1`** to **`a-=1`**
  - Mutation testing is an instance of this; can also just inject new code.
- Run-time injection: use software triggers (e.g., interrupts) to inject faults into running code
  - E.g., inject bad packets; corrupt memory

Example – SFI Whitebox

```
1 int max(int x, int y)
2 {
3 int mx = x;
4 if (x > y)
5     mx = x;
6 else
7     mx = y;
8 return mx;
9 }
```

```
1 int max(int x, int y)
2 {
3 int mx = x;
4 if (x < y)
5     mx = x;
6 else
7     mx = y;
8 return mx;
9 }
```

Fault Injection (Mutation) Operators

- Operand Replacement Operators:
  - Replace a single operand with another operand or constant. E.g., in (x>y)
    - if (5 > y) Replacing x by constant 5.
    - if (x > 5) Replacing y by constant 5.
    - if (y > x) Replacing x and y with each other.
  - E.g., if all operators are {+,-,*,**,/} then the following expression a = b * (c - d) will generate 8 mutants:
    - 4 by replacing *
    - 4 by replacing -.

Operators

- Expression Modification Operators:
  - Replace an operator or insert new operators. E.g.,
    - if (x == y)
    - if (x >= y) Replacing == by >=.
    - if (x == ++y) Inserting ++.

Operators

- Statement Modification Operators:
  - Delete the else part of an if-else statement.
  - Delete the entire if-else statement.
  - Replace line 3 by a return statement.

Assumptions

- *Competent Programmer Hypothesis:* Most software faults introduced by experienced programmers are due to small syntactic errors.
- *Coupling Hypothesis:* single faults can cascade to lead to more complex faults.

What fault could we add here?

```
/**
 * Adds a value to draw.
 *
 * @param x the x value.
 * @param y the y value
 */
public void addValue(double x, double y) {
    // if the maximum x is too low we extend it.
    if (maxX<x) {
        maxX=1.2*x;
        nDigitsX=(int)Math.floor(Math.log10(maxX));
    }
    // if the maximum y is too low, we extend it.
    if (maxY<y) {
        maxY=2*y;
        nDigitsY=(int)Math.floor(Math.log10(maxY));
        leftBorder = 20+nDigitsY*7;
    }
    // we add the point to the graph
    series[0].add(x);
    series[1].add(y);
}
```

Class Exercise – 3-5 minutes

- Sketch how you could apply each of the following testing types to PhonePicture:
  - Partition testing
  - Boundary testing
  - Coverage-based testing
  - Fault injection

PhonePicture – *win prizes* (also have strangers make rude comments about your photos)

- Any questions?

Section 3 – You Can Classify Testing Techniques

- Exploratory testing
- Specification-based testing
- Model-based testing
- Fuzz testing
- Partition testing
- Boundary testing
- Coverage-based testing
- Fault injection

Are they static or dynamic?

- Exploratory testing
- Specification-based testing
- Model-based testing
- Fuzz testing
- Partition testing
- Boundary testing
- Coverage-based testing
- Fault injection

Are they black/grey/white –box?

- Exploratory testing
- Specification-based testing
- Model-based testing
- Fuzz testing
- Partition testing
- Boundary testing
- Coverage-based testing
- Fault injection

Are they functional or non-functional?

- Exploratory testing
- Specification-based testing
- Model-based testing
- Fuzz testing
- Partition testing
- Boundary testing
- Coverage-based testing
- Fault injection

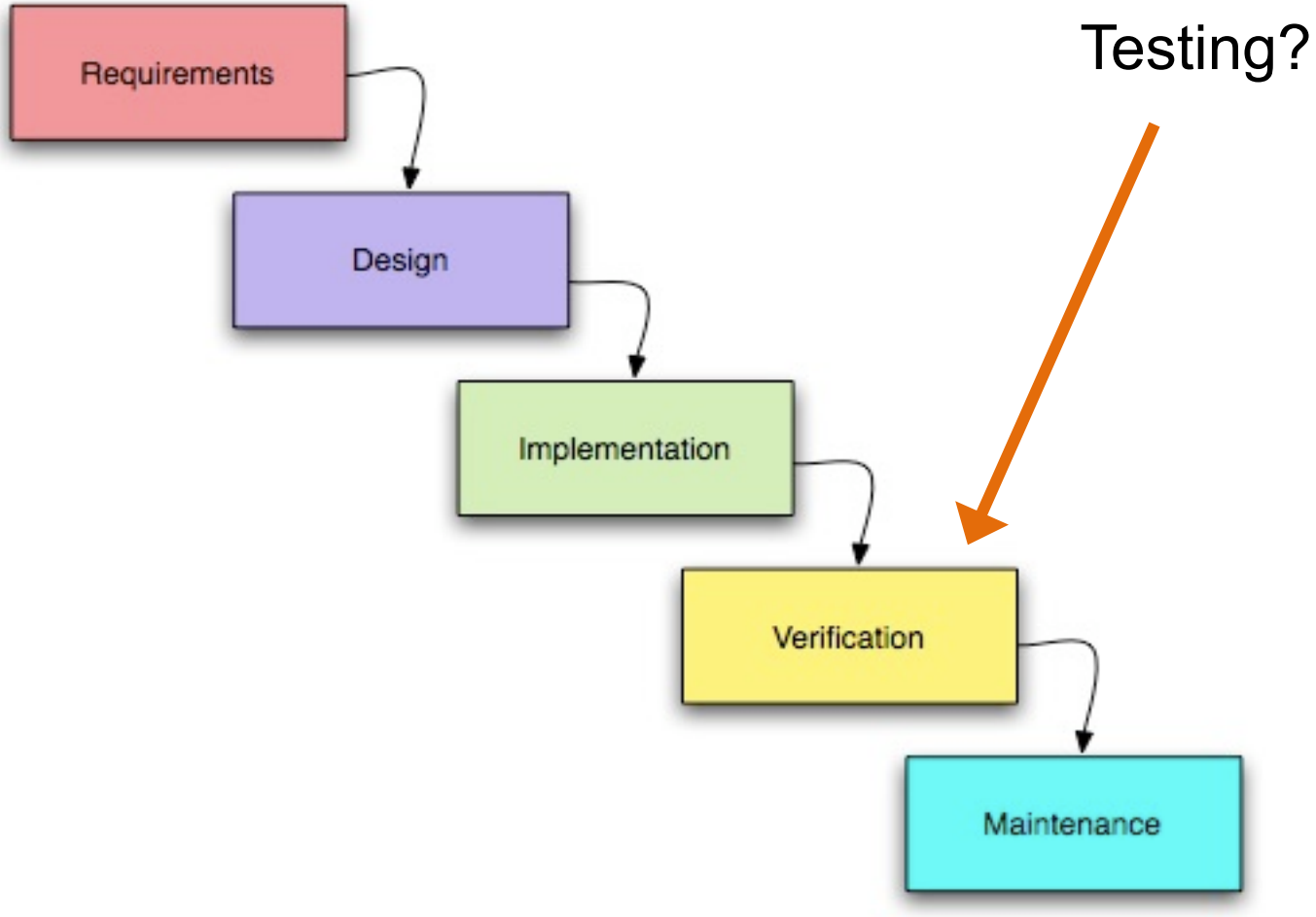
- Any questions?

Section 4 – You Can Fit Testing into Any Software Lifecycle

So, we have a number of useful testing techniques

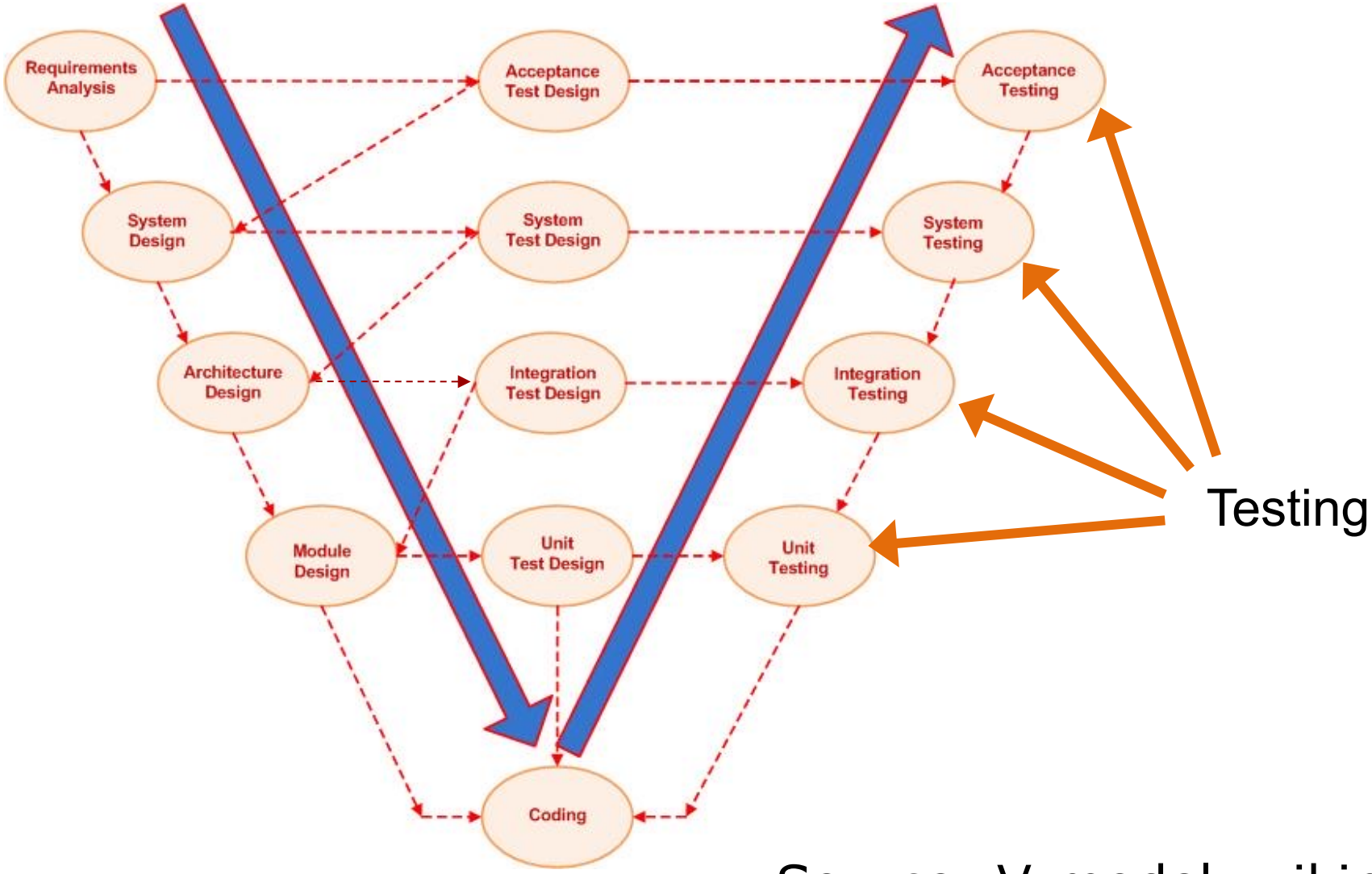
- But to use them, they have to fit into our software development somewhere
- And
  - Agile processes may test frequently, but without any specification
  - Waterfall-esque processes may want tests defined before coding
- Key thing is that *testing tasks* happen somewhere and somewhen

Waterfall model



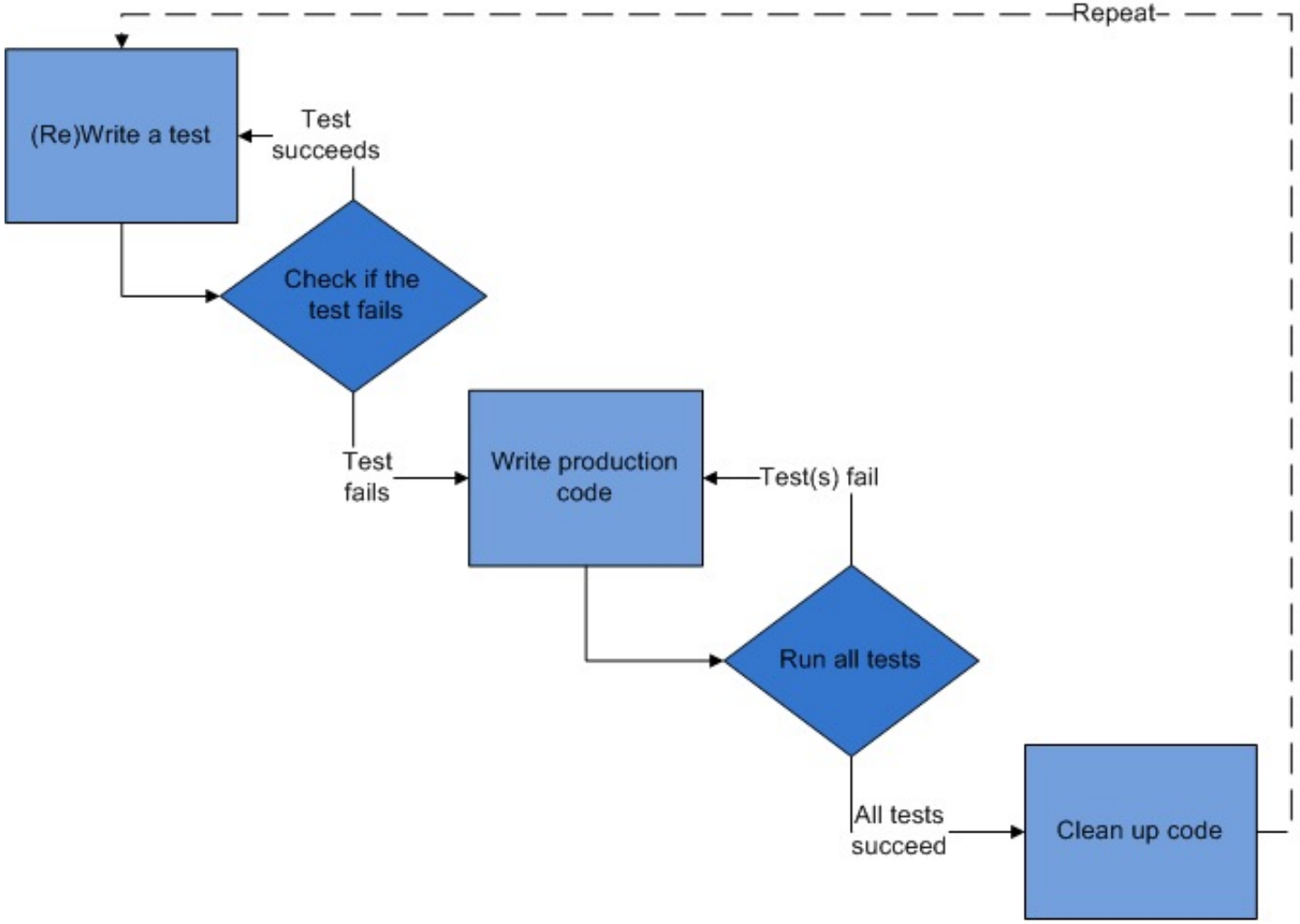
Winston Royce, 1970, source: wikipedia, waterfall model

V-Model



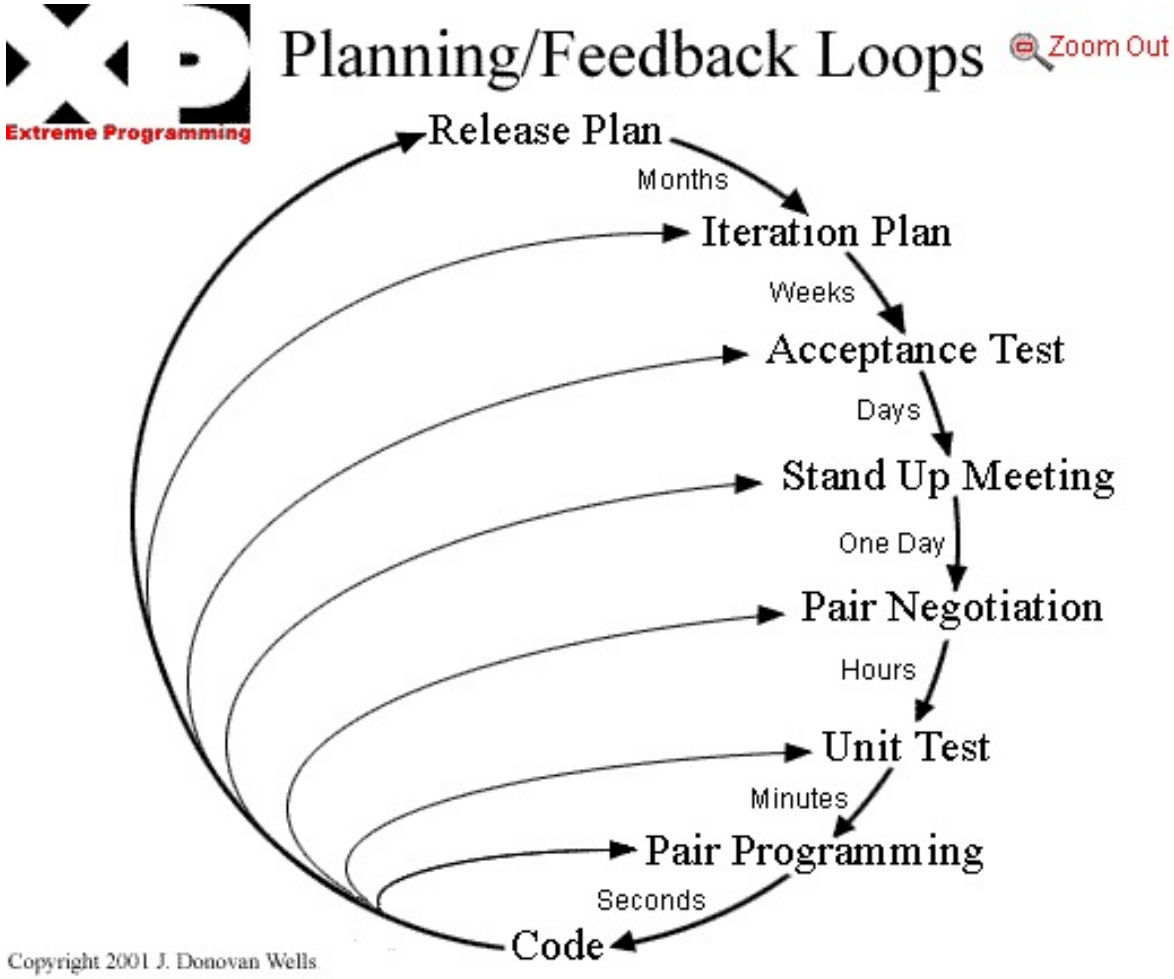
Source: V-model, wikipedia

Test-Driven Development



K. Beck (2003),
Source: Test-driven_development, wikipedia

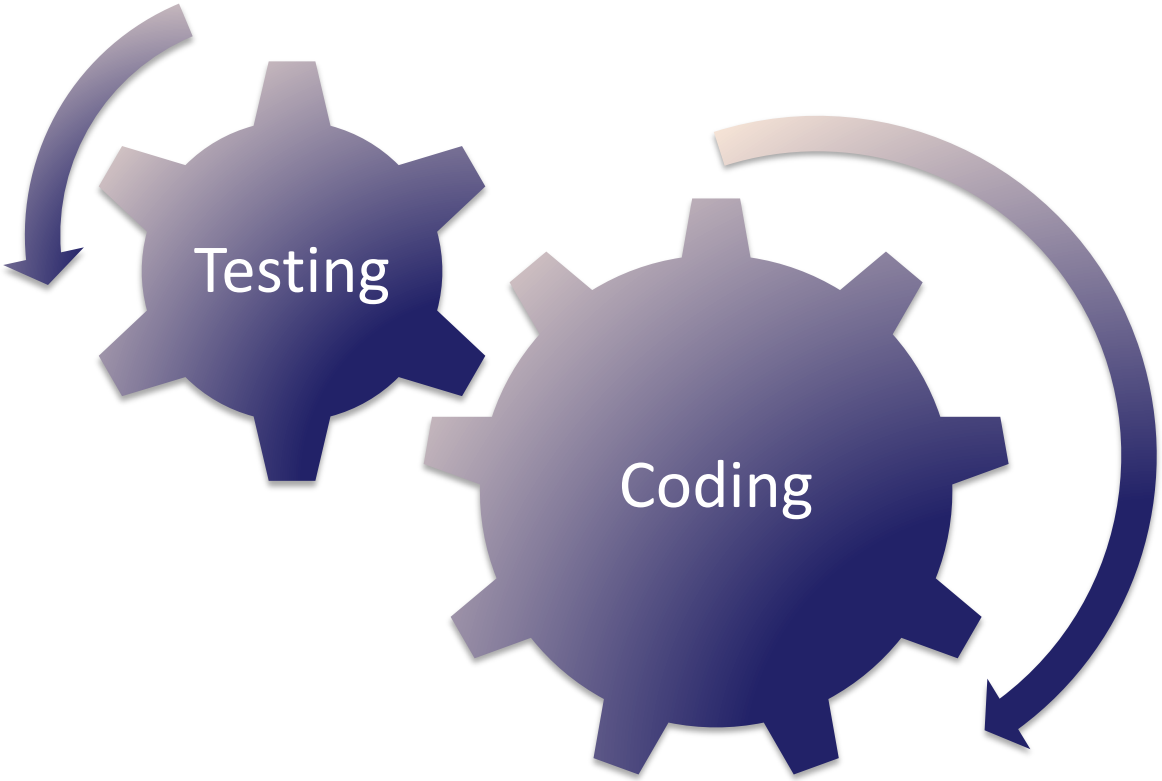
Extreme Programming



http://www.extremeprogramming.org/introduction.html

How is testing typically performed?

With modern development models…

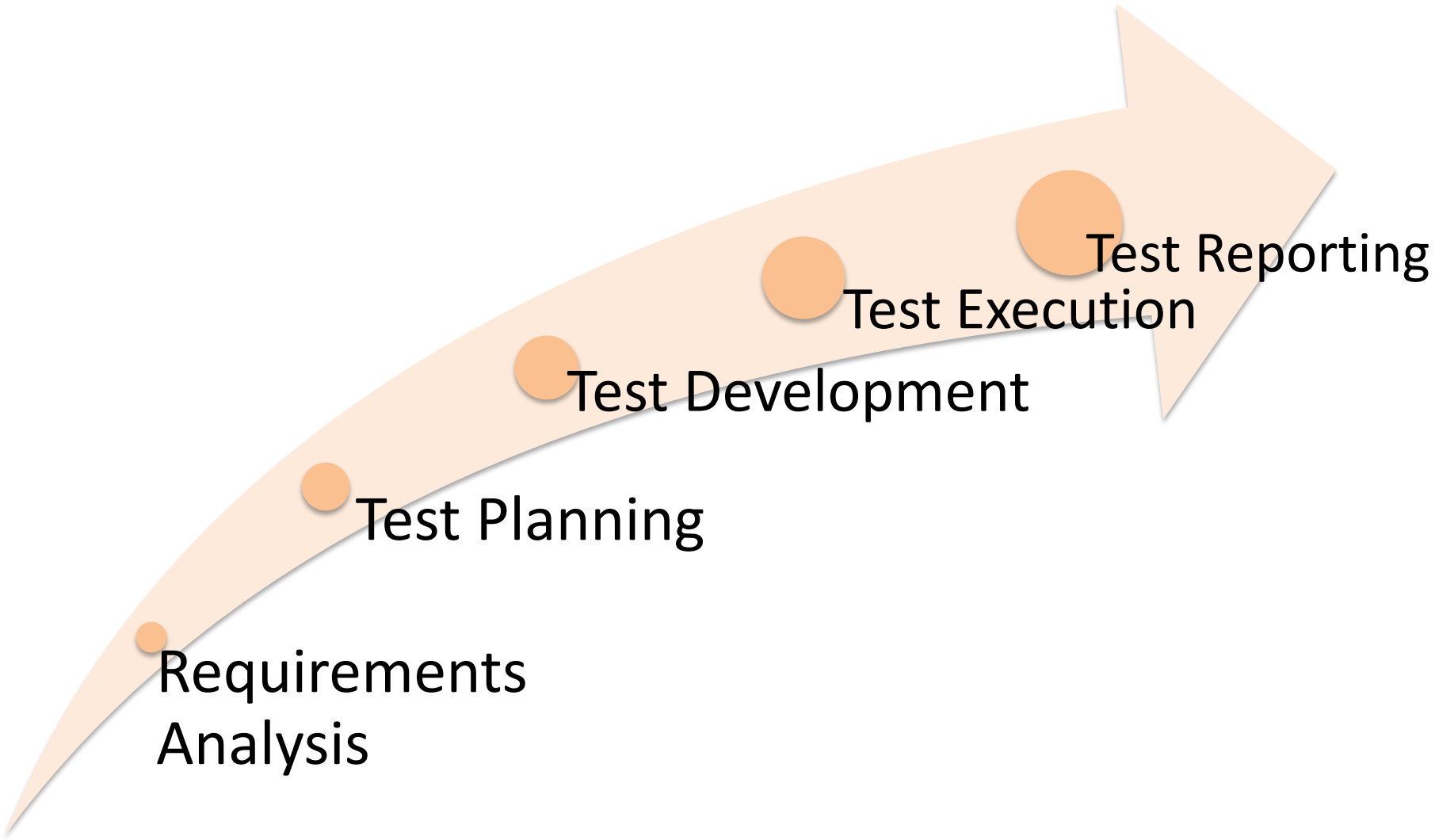


Some processes support testing better than others

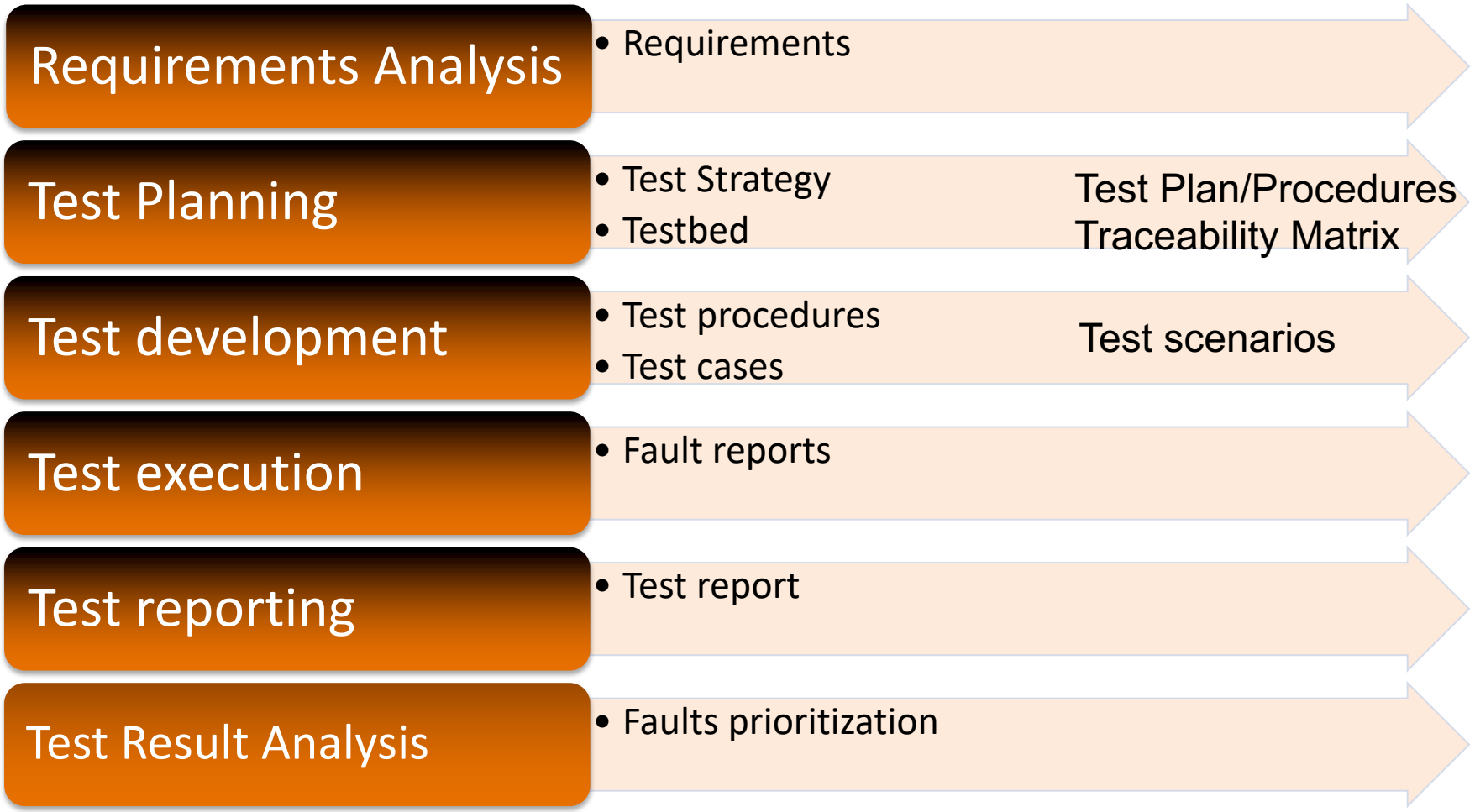
- E.g. pure waterfall doesn't let testing be much use

- That's a very good reason to not use those processes!

A typical process



Artifacts


Requirements Analysis
• Requirements
Test Planning
• Test Strategy
• Testbed
Test Plan/Procedures
Traceability Matrix
Test development
• Test procedures
• Test cases
Test scenarios
Test execution
• Fault reports
Test reporting
• Test report
Test Result Analysis
• Faults prioritization

Exercise – 3-5 minutes

- Assume you've got venture capital and 12 months to bring PhonePicture to market
- How might you plan to do different testing activities across this time?
- (Hint: split the time into 3 or 4 phases)
- (Hint: there are a lot of different sensible answers here)

PhonePicture – *send pictures to your friends* (also accidentally to your mom)

- Any questions?

Section 5 – Context-Driven Testing

Question – How do you know the right way to do testing?

Question – what is “context” here?

- “World” (in Spolsky’s sense)
- Stage of development
- Skills of test team
- Novelty of product
- Availability of reference models (e.g. previous versions)
- Availability and quality of explicit specifications
- ….

What if we can't do the tests we want?

- E.g. the regulator *insists* we do MC/DC testing on our avionics software
  - That's going to eat all the budget that we wanted for exploratory testing
- …*and* they won't let us make a prototype and fly it over the sea

So…

- Testing costs!
  - You have to weigh it against other things
- Exhaustive testing is almost never possible
- Effort alone is no guarantee
  - Must *target* that effort well

When should you *stop* testing?

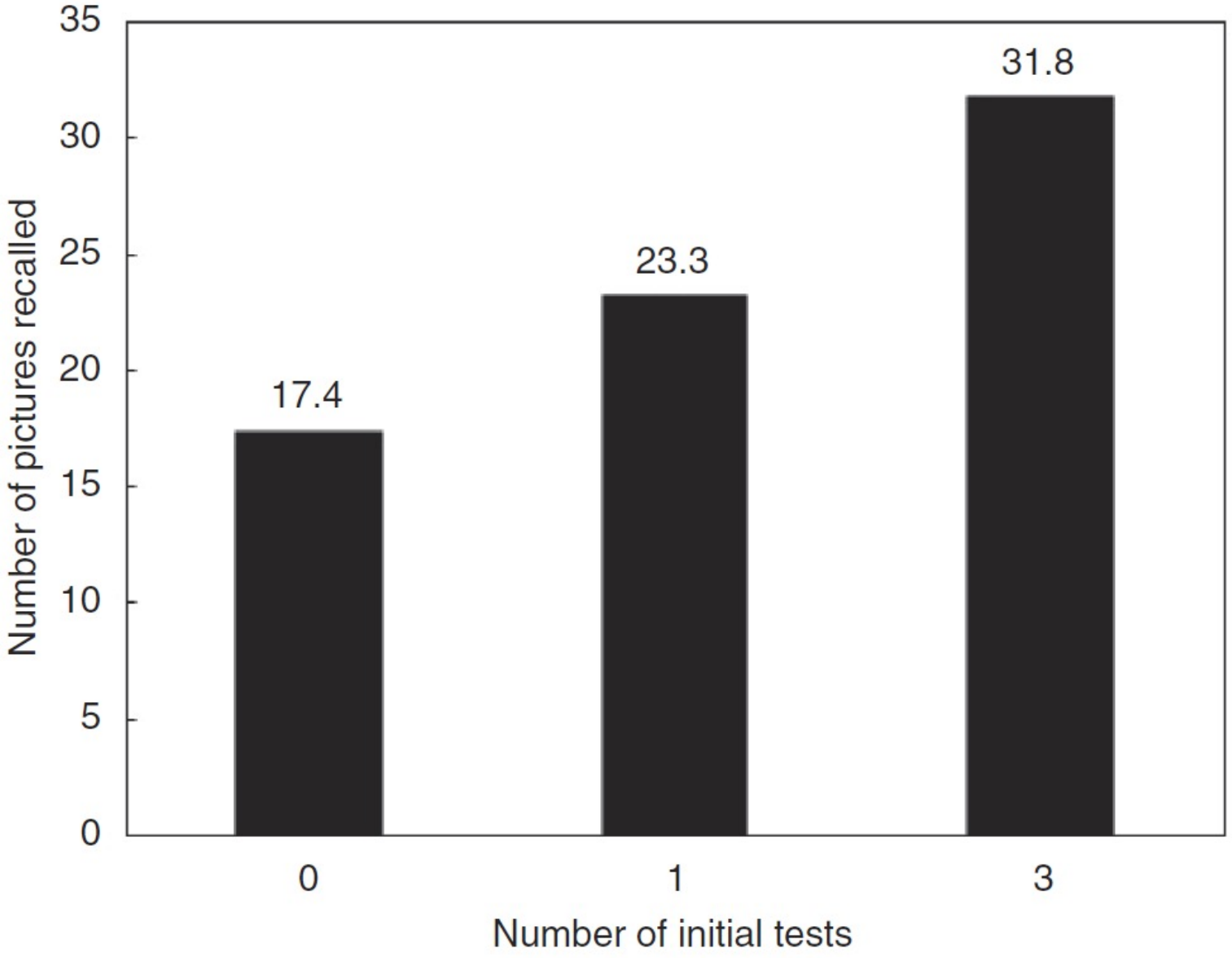
- Meet pre-agreed coverage goals
- Meet pre-agreed defect discovery rate
- Marginal cost
  - Estimated cost per defect discovered
    vs
  - Estimated cost of a typical defect (if product ships with it)
- Team consensus
- Management tells you to ship it

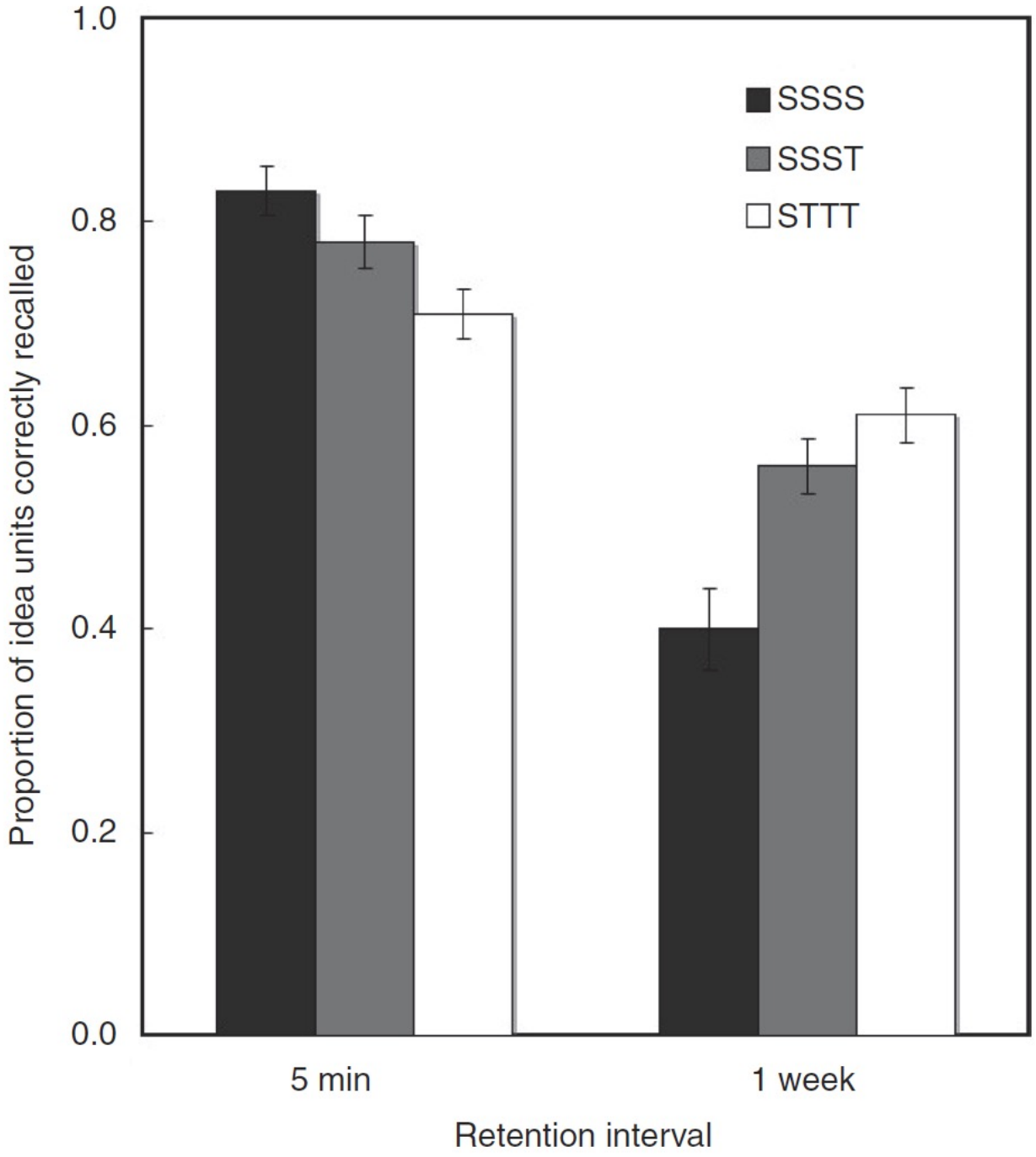
- Any questions?

Section 6 – That Quiz Again

The diagnostic quiz - repeated

- Why have I asked you to do this quiz again?
- Big increase in long-term recall
  - From (Jones 1923, Butler and Roediger 2007, Lyle and Crawford 2011) and many other studies


35
30
25
20
15
10
5
0
Number of pictures recalled
17.4
23.3
31.8
0
1
3
Number of initial tests


1.0
0.8
0.6
0.4
0.2
0.0
Proportion of idea units correctly recalled
SSSS
SSST
STTT
5 min
1 week
Retention interval

- Answer the quiz again
- Work alone

3-5 minutes

Diagnostic Quiz (Again!)

1. What are three kinds of software systems that each need very different testing?
2. What is “exploratory testing”?
3. What is “specification-based testing” aka “requirements-based testing”?
4. When you are fuzz testing (pouring random inputs into a program interface), you often can’t know in advance what the correct response to an input it is. How can it still be useful in that case?
5. What effect does a pure waterfall development lifecycle have on testing?
6. What’s a good reason to stop testing?

The diagnostic quiz – my answers

1 What are three types of software systems that each need very different testing?

Many possible answers, but a reasonable set would be:

- Mainstream shrinkwrap i.e. ordinary apps for lots of public users
- Organization-internal i.e. users are part of org that develop/supplies it, and are obliged to use it as part of their job etc
- Embedded i.e. runs on hardware inside a very small device that's not (in itself) a "computing device" e.g. not a computer, tablet or phone

2 What is "exploratory testing"

Manual testing without a strict test plan – exploring program features in a goal-directed way but no pre-committing to what you'll do.

The diagnostic quiz – my answers (2)

3 What is "specification-based testing" aka "requirements-based testing"?

Testing software against all the explicit claims made about it in the specification / requirements doc.

4 When you are fuzz testing (pouring random inputs into a program interface), you often can't know in advance what the correct response to an input it is. How can it still be useful in that case?

Sometimes it makes the program crash or otherwise do something obviously wrong. If not, it's often possible to define assertions for the bounds of correct behaviour (e.g. "The system must never do X") and check whether they're ever violated.

I.e. you don't have to know the exact right response to know that a given response is wrong.

The diagnostic quiz – my answers (3)

5 What effect does a pure waterfall development lifecycle have on testing?

It makes it very hard for it to be of much use, since you only do it at the end once you've done a huge amount of previous work.

6 What's a good reason to stop testing?

Many possible answers, but I would suggest defaulting to a combination of meeting basic coverage goals (statement coverage at the minimum) and getting consensus among the team.

- Any questions?

  Now

  By email

  (I'll answer by email, or in next lecture)

References

- H E Jones, The Effects of Examination on the Performance of Learning, Archives of Psychology 10, 1923

- Andrew C. Butler and Henry L. Roediger, III. Testing improves long-term retention in a simulated classroom setting. European Journal of Cognitive Psychology 19(4/5), 2007

- Keith B. Lyle, Nicole A. Crawford, Retrieving Essential Material at the End of Lectures Improves Performance on Statistics Exams. Teaching of Psychology 38(2), 2011