Testing

John D. McGregor
Dept of Computer Science
Clemson University

Levels of testing

Unit
- A fundamental unit, such as a class, is tested to determine whether it fulfills its post-conditions

Subsystem
- A set of integrated units are tested to be certain that they play nicely together

System
- A completed application is tested as it will be used

Types of testing

Functional
- Tests are based on the specification of WHAT the item does

Structural
- Tests are based on the structure that defines HOW the unit operates

Interaction
- Tests are based on the communication that happens between units

Selecting test cases

Input-based
- Analyze the data types of parameters and variables based on the problem domain
- Partition into "equivalence classes"
- Select inputs from each equivalence class
- *setTemperature(int degrees) if the domain refers to water, the equivalence classes might be degrees<32; 32<=degrees<=212;degrees>212*
- *We assume that any value between 32 and 212 will cause the program to use the same mechanism so if one value is computed correctly then all will be*

Output-based
- Based on possible outcomes
- Win game
- Lose game
- Select test data to achieve these outcomes

Test coverage

void doIt(int x)

If you select one value for x, execute doIt for that value, and get a correct answer, how confident are you that doIt works?

What if you run 50 values through and get correct answers? Feel better?

The more completely a domain is covered by test cases, the more likely the result is to be correct.

Test process

Testing operates in parallel to the development process

Unit tests are written and applied by the developer
They are applied when the developer is ready

Subsystem tests are often written and applied by an "integration" team
These integrations or "builds" are scheduled when all the classes in a subsystem are ready

System tests are written and applied by a system test team.
Scheduled when sufficient end-user functionality is available to test

Unit tests – functional tests

Based on the public interface of the unit

public int largest(int a, int b, int c)

Test cases sample from the "domain" of the input Possible inputs are divided into equivalence classes

a	b	c	correct answer
1	3	5	5
1	5	3	5
5	1	3	5
5	3	1	5

Example

>javac –classpath .;%classpath% trial5.java

>java –classpath .;%classpath% trial5

5

Repeat this using all the tests on the previous page

Are we confident this method is correct?

Here is the code

```
public class trial5{
     public trial5(){
                }
     public int largest(int a,int b,int c){
           return 5;
     }

     public static void main(String[]  args){
           trial5 t = new trial5();
           System.out.println(t.largest(1,3,5));
     }
}
```


Tests from OCL

Test cases must establish the pre-conditions

Test cases are developed using the specified outcomes in the post-condition

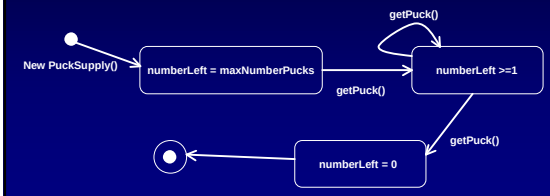
Puck PuckSupply::getPuck()

pre: true

post: result = pucks->asSequence->first and size = size@pre - 1

Tests from UML

Tests are sequenced to "cover" the transitions



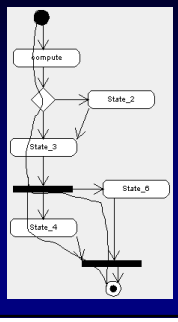
Unit tests - structural

Based on the structure of the code

```
Public int largest(int a, int b, int c)
{
     if (a > b){
          if(a > c){
               return a;
          }
          else{
               if(c > b){
                    return c;
               }
          }
     else return 0;
}
```

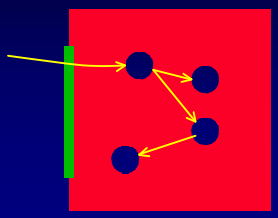
Tests from UML

Paths through the activity diagrams give a set of test cases



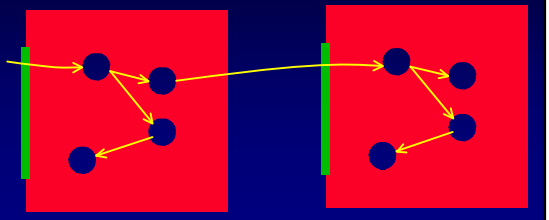
Subsystem tests

In object-oriented software, integrated units may still be a single object:

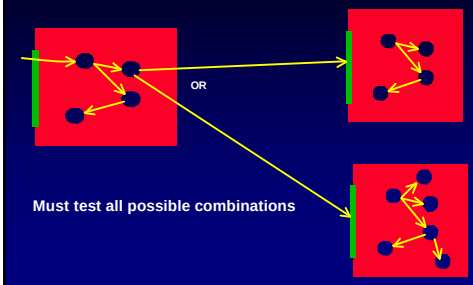


Subsystem

The two objects may be in different processes.



Polymorphic substitution



Must test all possible combinations

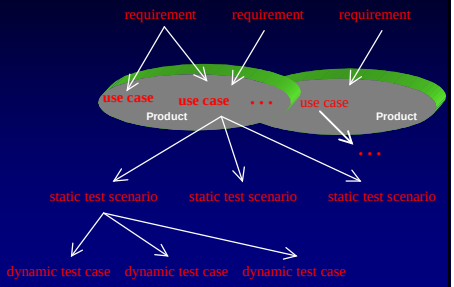
System tests

Predominantly functional

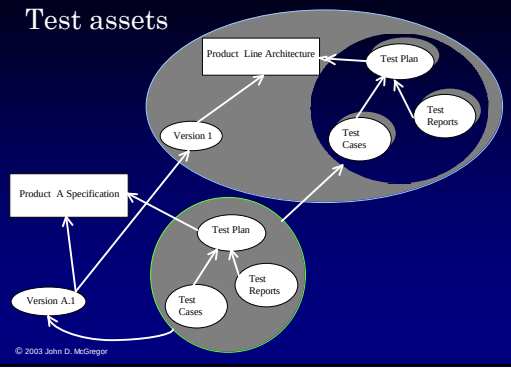
Based on use cases

Additional test cases are targeted at specific non-functional requirements such as performance, security, …

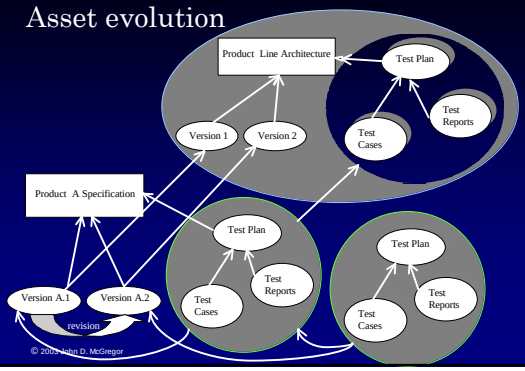
The use case hierarchy



Test assets



Asset evolution



System tests - Brickles

Inputs: mouse – where can it be moved

Test cases:

Input-based

- **Paddle hits puck on first descent**
- **Paddle allows puck to hit floor**

Output-based

- **Paddle hits puck several times in a row but then loses it and eventually loses all pucks – loss**
- **Paddle hits puck many times – eventually game is won**

Software wears

Just like physical objects, software wears overtime
Changes in DLLs and operating systems lead to this wear
This leads to the need for self-tests

Additional tests

Performance tests

Security tests

Certification tests

Acceptance tests

Deployment tests

Combinations of variables

Suppose we have 3 variables
And that each variable can take on 3 values

27 possible combinations

1,1,1
1,1,2
1,1,3
1,2,1
1,2,2
1,2,3
…

Pair-wise combinations

Factor 1	Factor 2	Factor 3
1	1	3
2	1	2
3	1	1
1	2	2
2	2	1
3	2	3
1	3	1
2	3	3
3	3	2

Application

Handset Manufacturer	Protocol Set	Coverage Area
A	GPRS	international
B	GPRS	national
C	GPRS	local
A	EGPRS	national
B	EGPRS	local
C	EGPRS	international
A	UMTS	local
B	UMTS	international
C	UMTS	national