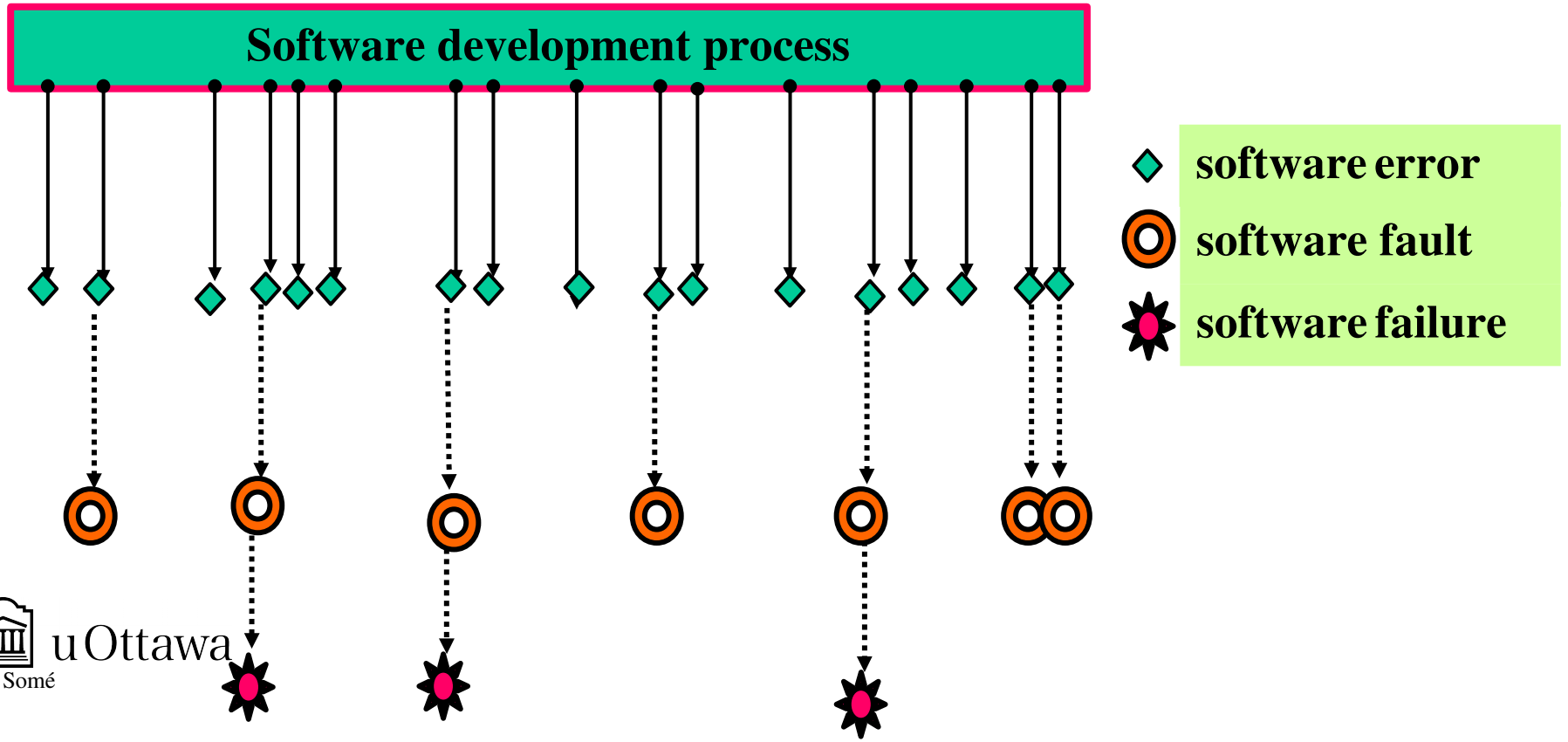
Software Testing

Introduction

What is Software testing ?

•Examination of a software unit, several integrated software units or an entire software package by running it.

- execution based on *test cases*
- expectation – reveal faults as failures



Objectives of testing

- To find defects before they cause a production system to fail.
- To bring the tested software, after correction of the identified defects and retesting, to an acceptable level of quality.
- To perform the required tests efficiently and effectively, within budgetary and scheduling limitation.
- To compile a record of software errors for use in error prevention (by corrective and preventive actions)

Testing Axioms

- It's impossible to test a program completely
  - large input space
  - large output space
  - large state space
  - large number of possible execution paths
  - subjectivity of specifications

Testing Axioms

Large Input/State space

```
int exFunction(int x, int y) {
  ...
}
```

- Exhaustive testing ⇒ trying all possible combination of *x* an *y*

Testing Axioms

- Large number of possible execution paths

Number of paths = 2^n

```
...
for (int i = 0; i < n; ++i) {
  if (a.get(i) == b.get(i))
    x[i] = x[i] + 100;
  else
    x[i] = x[i]/2;
}
...
```

N	Number of paths
1	2
2	4
3	8
10	1024
20	1048576
60	1.15E+18

With 10^{-3} sec per test case
need since big-bang for n = 36

Testing Axiom

- Large number of states

```
class Account {
  private:
   AccountNumber number;     // 6 digit account id
   Money         balance;    // current balance
   Date          lastUpdate; // days since last transaction
   ...
}
```

- How many states?

Testing Axioms

- Upper limit to total number of tests (Binder)

$$(2^n \times (L_1 \times L_2 \times \cdots \times L_X) \times (V_1 \times V_2 \times \cdots \times V_Y))!$$

  - n number of decisions
  - Li number of times a decision can loop
  - X number of decisions that cause loops
  - Vi number of all the possible values each input variable could have
  - Y number of input variables.

Testing Axioms

- Software testing is a risk-based exercice
  - need to balance cost and risk of missing defects
- Testing can't prove the absence of defects
- The more defects you find, the more defects there are likely
- The pesticide paradox
  - a system tends to build resistance to a particular testing technique

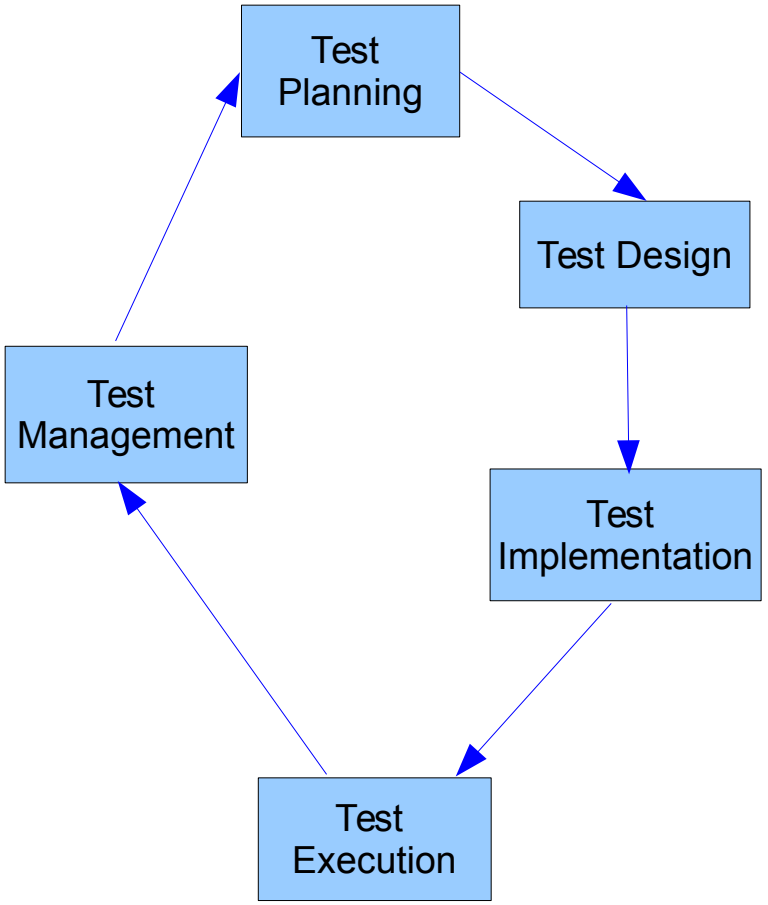
Testing Axioms

- Not all defects found are fixed
- Defects are not always obvious
  - only when observed (latent otherwise)
- Product specifications are never final
- Software testers aren't the most popular members of a project team
- Software testing is a disciplined technical profession that requires training

What Testing can accomplish ?

- Reveal faults that would be too costly or impossible to find using other techniques
- Show the system complies with its stated requirements for a given test suite
- Testing is made easier and more effective by good software engineering practices

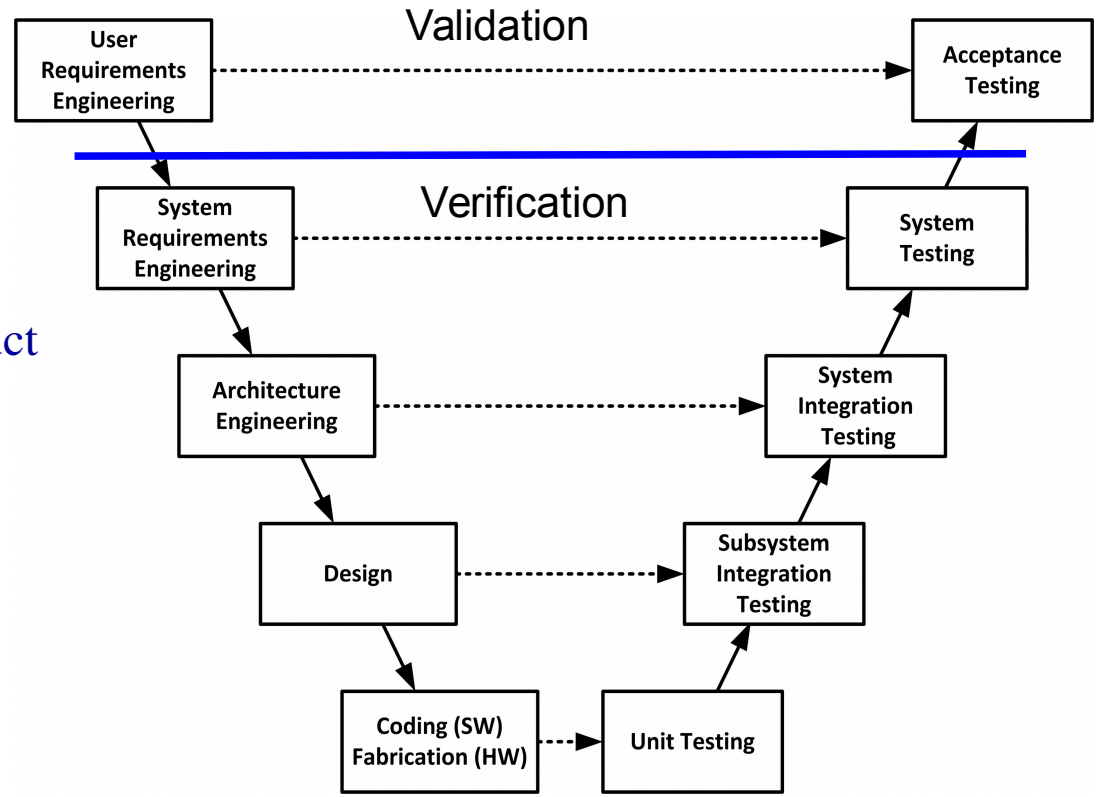
Software Testing Process



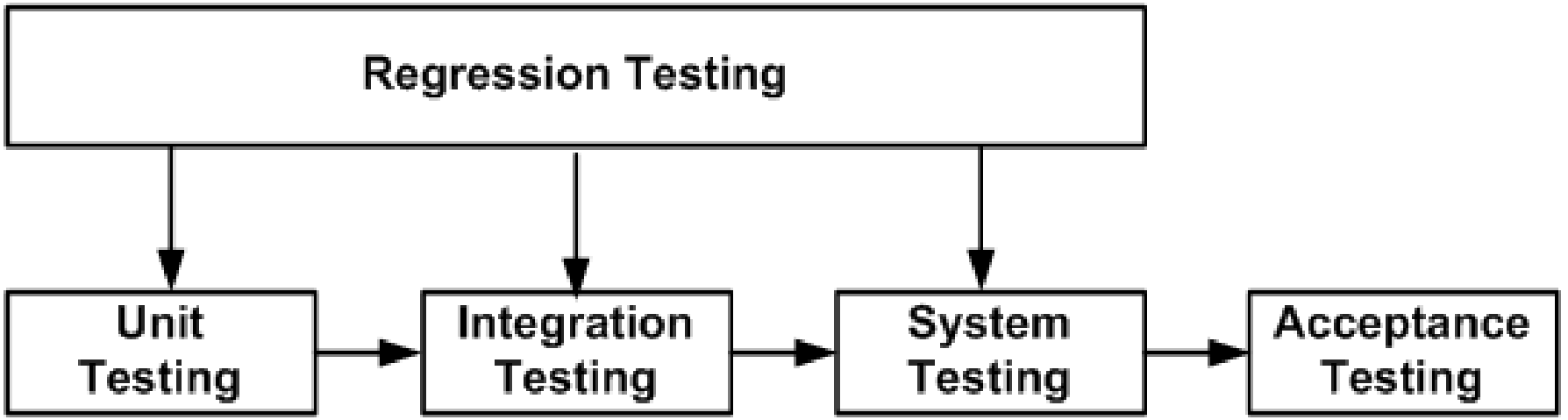
- **Planning**
  includes completion criteria (coverage goal)
- **Design** - approaches for test case selection to achieve coverage goal
- **Implementation** – scripting of test cases
  - input/output data
  - state before/after
  - test procedure
- **Execution** – run tests
  - check results – pass or fail?
  - coverage ?
- **Test Management** – defect tracking, maintain relationships, etc.

Levels of Testing

- Unit testing
  - Individual program units, such as procedure, methods in isolation
- Integration testing
  - Modules are assembled to construct larger subsystem and tested
- System testing
  - Includes wide spectrum of testing such as functionality, and load
- Acceptance testing
  - Customer's expectations from the system



Levels of Testing



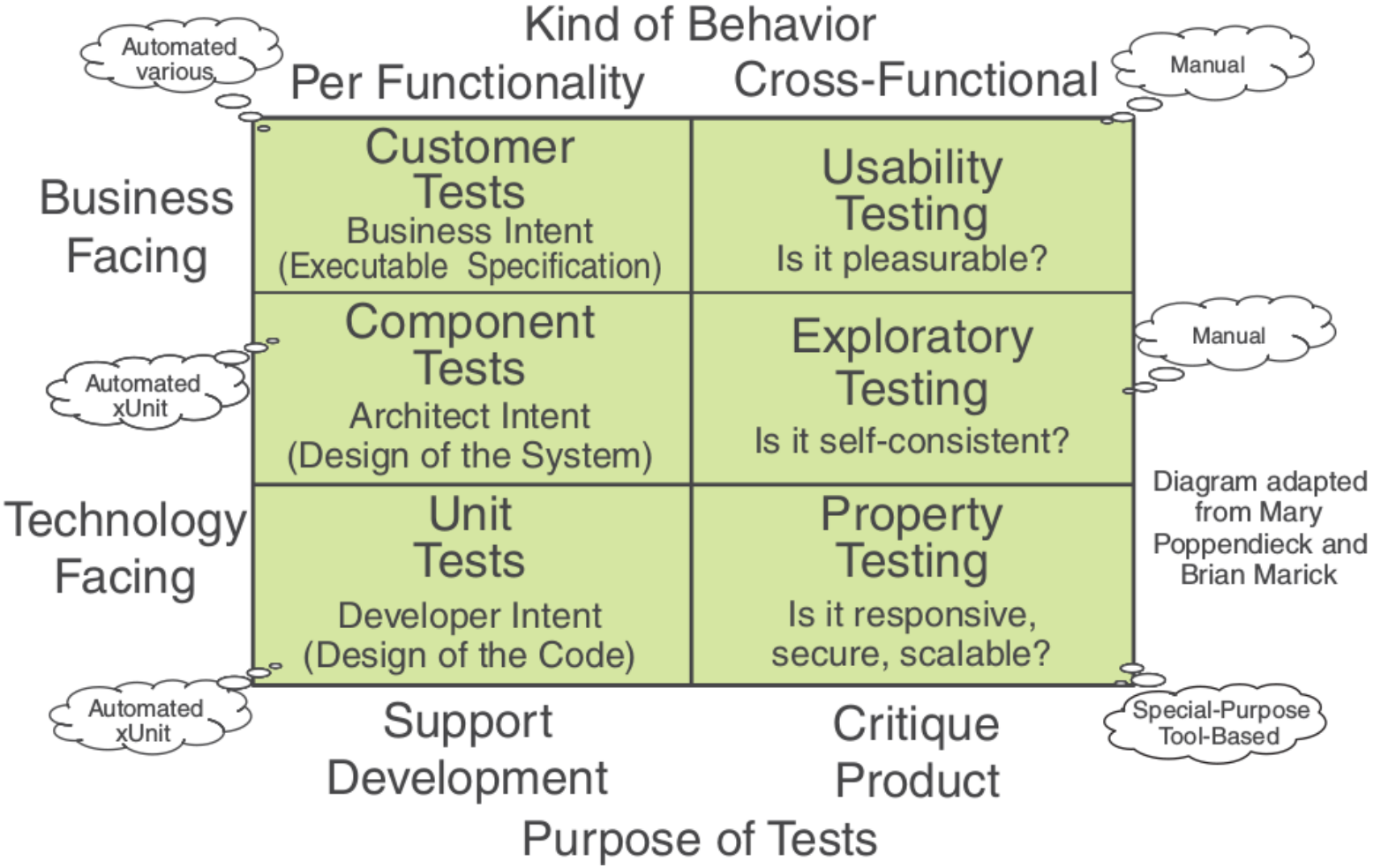
- After updates
  - tests are selected, prioritized and executed
  - to ensure that changes do not introduce defects

Test Design Approaches

- Black Box testing
  - tests design based on specification (input/output)
- White Box testing
  - tests design based on code structure
- Grey Box testing
  - tests design based on design model

Agile Testing

- Agile Testing Matrix (Meszaros)



Agile Testing

- Customer tests (functional tests, acceptance tests, end-user tests)
  - verify the behavior of the application from the point of view of Customers
  - developed based on requirements (e.g. user stories) by specifying acceptance criteria
  - developed in collaboration by Customers, Users, Testers, Developers
  - written in a language understandable by Customers

Agile Testing

- Unit tests
  - verify the behavior of a program unit (e.g. single class, method, function) that is a consequence of a design decision
  - written by developers
  - summarize the behavior of the unit in the form of tests

Agile Testing

- Component tests (integration tests)
  - verify components consisting of groups of units that collectively provide some service
  - written by developers (derived from customer tests)

Agile Testing

- Property tests (nonfunctional, cross-functional tests)
  - verify nonfunctional requirements (e.g. response time, capacity, stress, security, ...)
  - automation (beyond xUnit tools) is essential for most of these tests
  - tests must start as soon as possible (basic architecture, skeleton of functionality) and reexecuted continuously

Agile Testing

- Usability tests
  - verify fitness for purpose
  - can real users use the software application to achieve the stated goals ?

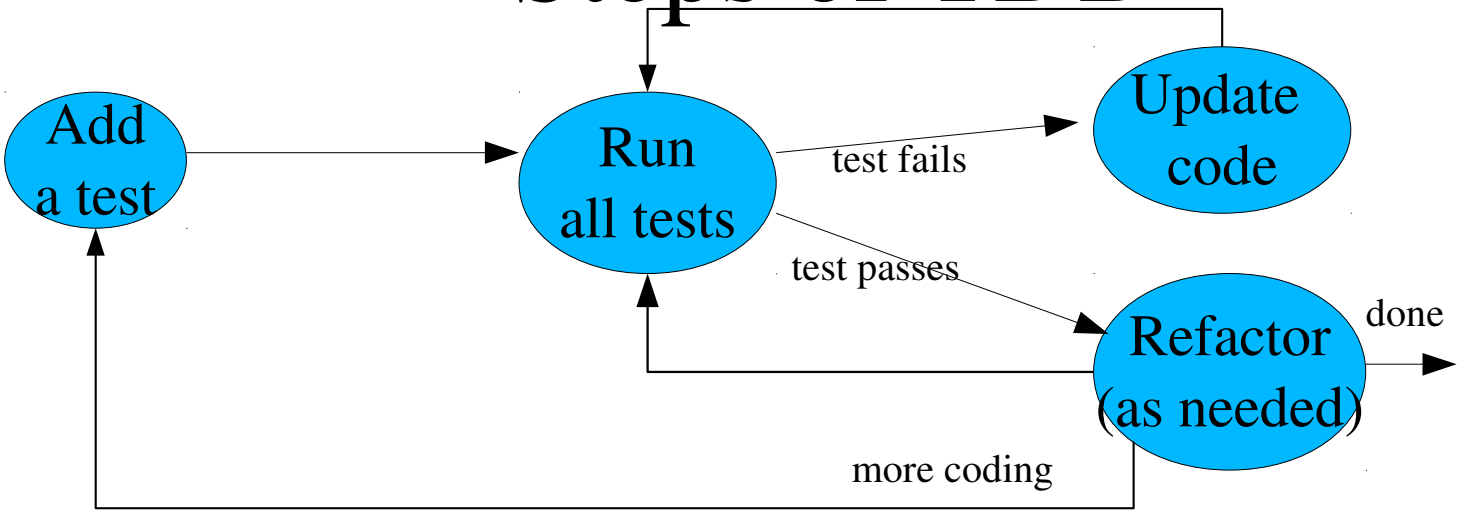
Agile Testing

- Exploratory tests
  - to determine whether the product is self-consistent
  - testers use the product, observe behavior, form hypotheses, design tests to very hypotheses, and exercise product with them

Test Driven Development (TDD)

- Software development approach based on writing tests first
  - test-first development
  - core technique for agile software development
  - several tests, executed often
    - automated tool (xUnit) is essential

Steps of TDD



1. Quickly add a test
   - the simplest test corresponding to a small piece of functionality
2. Run all the tests (or subset of targeted tests including the new test)
   - first run ensures that the new test fails
3. Update the functional code
   - simplest change so that it passes the new test
4. Repeat from step 2 until all tests pass
5. Refactor the code as needed
6. Repeat from step 1 if the coding is not done

Acceptance Test Driven Development (ATDD)

- Development starts with writing of an acceptance test
  - expected system behaviour from users' point of view
  - free of technical (implementation) details
  - written in collaboration by domain experts, customers, users, developers
  - expressed in simple understandable language

Testing Automation

- Use of automated tools is essential
  - provide speed, efficiency, accuracy, precision, etc
  - allow repeatibility (regression testing)
- Types of tools:
  - viewers and monitors (e.g. code coverage tool, debugger)
  - drivers and stubs
  - stress and load tools
  - analysis tools (e.g. file comparison, screen capture and comparison)
  - random testing tools (monkeys)
  - defect tracking

Types of Testing Tools
Test Planning and Management

- Create/maintain test plans
  - integrate with project plan
- Maintain links to Requirements/Specification
  - generate Requirements Test Matrix
- Reports and Metrics on test case execution
- Tracking of history/status of test cases
  - defect tracking

Types of Testing Tools
Test Design & Implementation

- Automated creation of test cases
  - Based on test design approaches
    - graph based
    - data flow analysis
    - logic based
    - ...
  - Combination generators
- Random test data generator
- Stubs/Mocks

Types of Testing Tools
Test Execution

- Test Drivers and Execution Frameworks
  - Run test scripts and report result
  - e.g. JUnit
- Runtime test execution assistance
  - comparators

Types of Testing Tools
Test Performance assessment

- Analysis of the effectiveness of test cases for extent of system covered
  - coverage analyzers
  - report on various levels of coverage
- Analysis of the effectiveness of test cases for defects detection
  - mutation testing

Types of Testing Tools
Specialized testing

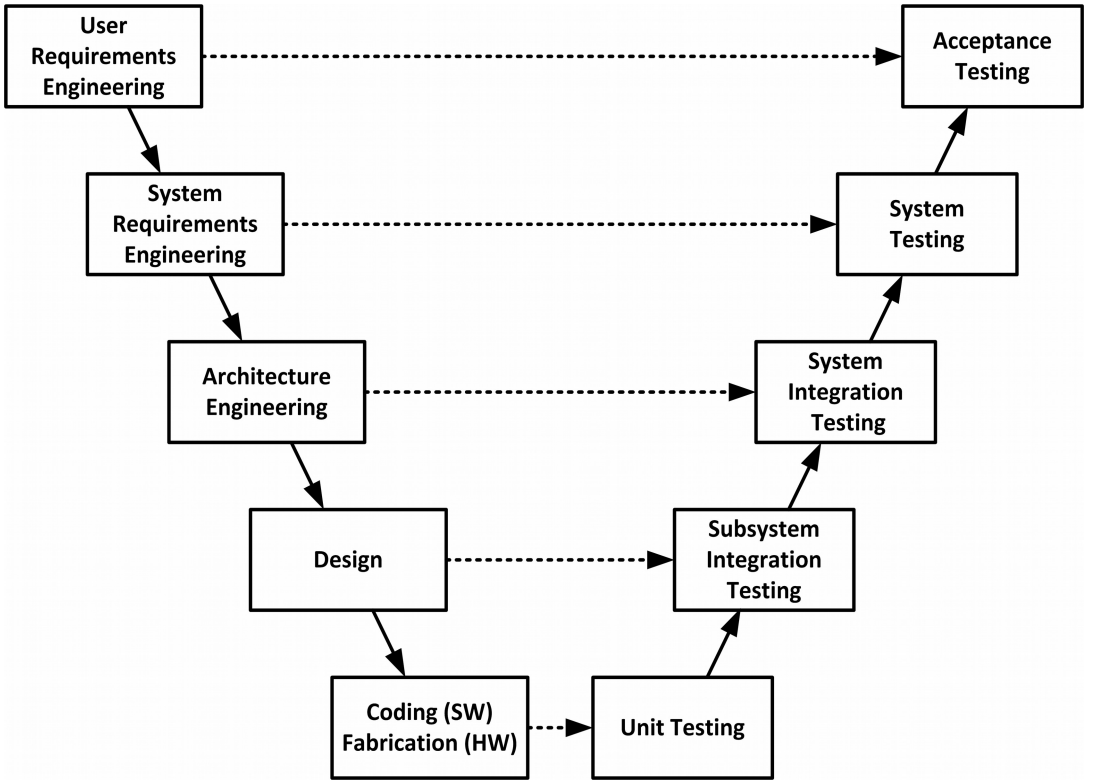
- Security testing tools
  - password crackers
  - vulnerability scanners
  - packet crafters
  - ...
- Performance / Load testing tools
  - performance monitors
  - load generators
  - ...

Types of Test Tools
Capture and Replay

- For testing from user interface (GUI, Web)
- *Records* a manual test session in a script
  - user inputs and "capture" of system responses
- Then, "*plays back*" the recorded user input and checks if the same responses are detected as are stored in the captured script.
- Benefits: relatively simple approach, easy to use, little/no scripting involved

Tool support at different levels

- Unit testing
  - Tools such as JUnit
- Integration testing
  - Stubs, mocks
- System testing
  - Functional, security, performance, load testers
- Regression testing
  - Test Management tools (e.g. defect tracking, ...)



What do we need to do automated testing?

- Test script
  - Test Case Specification
    - Actions to send to system under test (SUT).
    - Responses expected from SUT.
  - How to determine whether a test was successful or not?
- Test execution system
  - Mechanism to read test script, and connect test case to SUT.
  - Directed by a test controller.

What is a Test Case ?

- For a state-less system (outcome depends solely on the current input)
  - a pair of **<input, expected outcome>**
- For a state-oriented system (outcome depends both on the current state of the system and the current input)
  - a sequence of **<input, expected outcome>**

  ATM example:

  < check balance, $500.00 >,

  < withdraw, "amount?" >,

  < $200.00, "$200.00" >,

  < check balance, $300.00 >
- Various ways **input** may be specified

Expected Outcome

- An outcome of program execution may include
  - Value produced by the program
  - State Change
  - Sequence of values which must be interpreted together for the outcome to be valid

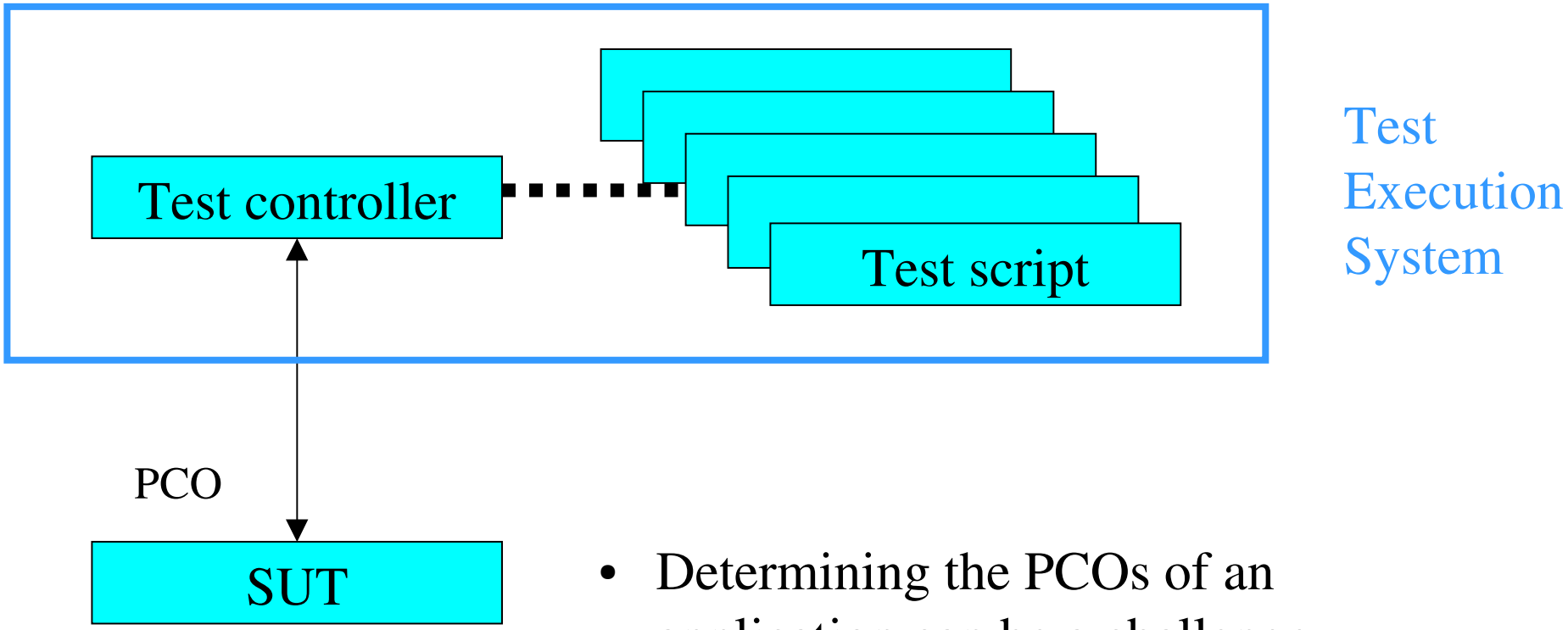
Expected Outcome

- Determination of expected outcome not always straightforward

- *Test oracle* - a mechanism that verifies the correctness of program outputs
  - Generates expected results for the test inputs
  - Compares the expected results with the actual results of execution of the SUT

Test Architecture

- Includes defining the set of Points of Control and Observation (PCOs)



- Determining the PCOs of an application can be a challenge.

Potential PCOs

- Some Potential PCOs:
  - Direct method call (e.g. JUnit)
  - User input / output
  - Data file input / output
  - Network ports / interfaces
  - Windows registry / configuration files
  - Log files
  - Pipes / shared memory

Potential PCOs

- 3rd party component interfaces:
  - Lookup facilities:
    - network: Domain Name Service (DNS), Lightweight Directory Access Protocol (LDAP), etc.
    - local / server: database lookup, Java Naming and Directory Interface (JNDI), etc.
  - Calls to:
    - remote methods (e.g. RPC, Services)
    - Operating System

- For the purposes of security testing, all of these PCOs could be a point of attack.