Software Engineering

Lecture 9

Lectures	Topics
1	Introduction to Software Engineering
2	Software Development Process (SDLC Activities) - SDLC Activity: Specification or Requirement Engineering - SDLC Activity: System Modeling/Design - SDLC Activity: Implementation - SDLC Activity: Testing - SDLC Activity: Evolution (-) - SDLC Activity: Deployment/Installation - SDLC Activity: Maintenance (-)
3	SDLC Activity: Requirement Engineering - Requirement Elicitation - Requirement Analysis and Management - Requirement Validation
4, 5, 6	SDLC Activity: System Modeling/Design
7,8	SDLC Activity: Implementation (GIT – Version management, IDE)
9,10	SDLC Activity: Testing + Maintenance
11	SDLC Activity: Deployment (cloud computing + Trends)
12	More on System Architecture

7.9.2

Black Box Testing

- **Black box testing methods focus on the functional requirements of the software**,
- i.e. derives sets of **input conditions** that will fully exercise all functional requirements for a program.
- Black box or functional testing is **based upon the specification of a module** rather than the implementation of the module.
    - Determines **whether the required functionality is present**, but since it is independent of the actual program code it cannot test parts of the program that are not covered by the specification.

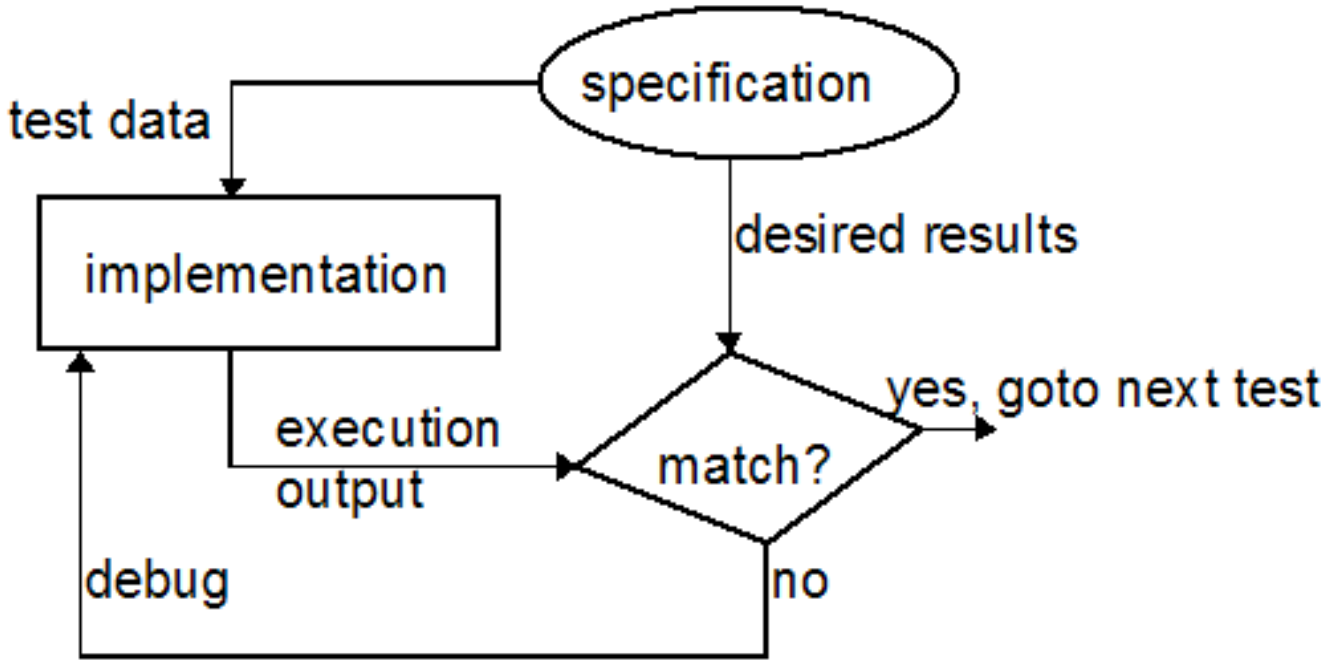
7.9.2

Black Box Testing

Black box testing attempts to find errors in the following categories:

1. Incorrect or missing functions
2. Interface errors
3. Errors in data structures or external databases access
4. Performance errors
5. Initialization and termination errors.

7.9.2 Black Box Testing



7.9.2.1

Equivalence Partitioning

- Divides the input domain of a program into classes of data.

- Equivalence classes may be defined according to the following guidelines:

1. If an input condition **specifies a range**, one valid and two invalid equivalence classes are defined.
   - E.g. min 5, max 5 character
2. If an input condition requires a **specific value**, one valid and two invalid equivalence classes are defined.
   - E.g. VAT Rate

7.9.2.1

Equivalence Partitioning

3. If an input condition specifies a **member of a set**, one valid and one invalid class are defined
   - E.g. Predefined list of category of item
4. If an input condition is **Boolean**, one valid and one invalid class are defined.
   - E.g. gender case, true – false case

7.9.2.1

Equivalence Partitioning: Range Condition

- Input Data variable is ***Employment Age***, as used in a company database system. Employment age is defined as an integer variable, and has an acceptable range of 16 to 65 (years old).

- Since this is a range, there will be one valid, and two invalid equivalence classes.

- **The valid class**: Employment age = from 16 to 65, inclusive. An example of a test case falling into this class would be Employment age = 30.

- **The first invalid class**: Employment age < 16. An example of a test case falling into this class would be Employment age = 10.

- **The second invalid class**: Employment age > 65. An example of a test case falling into this class would be Employment age = 80.

7.9.2.2 Boundary Value Analysis

- **Boundary value analysis** (BVA) select test cases at the "edges", and thus exercises bounding values.

- BVA leads to the selection of test cases at the “edges” of the class.

7.9.2.2

Boundary Value Analysis

- Input Data variable is ***Employment Age***, as used in a company database system. Employment age is defined as an integer variable, and has an acceptable range of 16 to 65 (years old).
- The two “edges” are 16 and 65.
- Hence, the six test cases would be:
- Employment Age = 15, 16, 17, and 64, 65, and 66.

7.9.2.3

Comparison Testing

- Used when the reliability of software is absolutely critical.
- **Each version is tested with the same test data to ensure that all provide identical output.**
- **Then all the versions are executed in parallel with a real-time comparison of results to ensure consistency.**
    - If the output from each version is the same, then it is assumed that all implementations are correct.
    - If the output is different, each of the applications is investigated to determine if a defect in one or more versions is responsible for the difference.

Automated Testing Tools

1. Code auditors
2. Assertion processors
3. Test file generators
4. Test data generators
5. Test verifiers
6. Output comparators
7. Modern tools:
   https://cdn-images-1.medium.com/max/1400/1*Q_IDqjKORWzoXmxvb0xEaw.png

```php
/*
 * When validation fails,
 * it should generate error messages,
 * and redirect to register page
 * */
public function testRegisterFails(){

    // Setting stage for failed validation
    // By putting f_name empty
    $input = [
        'first_name'            => '',
        'last_name'             => 'thapa',
        'email'                 => 'nirmal@gmail.com',
        'password'              => 'computer',
        'password_confirmation' => 'computer'
    ];

    Input::replace($input);

    $this->call('POST', 'register-student', $input);

    $this->assertRedirectedToRoute('register-student');
    $this->assertSessionHas('errorMessage');
    $this->assertSessionHasErrors();
}
```

Example of automated test using PHPUnit Testing Framework

Example of automated test using PHPUnit Testing Framework

```
// the url, 'register'should create register page
public function testRegistrationFormView(){
    $this->call('GET', 'register-student');

    $this->assertResponseOk();
}
```

```php
/*
 * When validation passes,
 * it should generate error messages,
 * and redirect to register page
 * */
public function testRegisterSuccess(){
    // Setting stage for successfull validation
    // By putting correct values
    $input = [
        'first_name'            => 'nirmal',
        'last_name'             => 'thapa',
        'email'                 => 'nirmal@gmail.com',
        'password'              => 'computer',
        'password_confirmation' => 'computer'
    ];

    Input::replace($input);

    $this->call('POST', 'register-student', $input);

    $this->assertRedirectedToRoute('register-student');
    $this->assertSessionHas('successMessage');
}
```

Example of automated test using PHPUnit Testing Framework

```php
public function testLoginSuccess(){

    $this->markTestIncomplete();

    // Setting stage for successfull login
    // By putting correct values

    $input = [
        'email'                 => 'xyz@gmail.com',
        'password'              => 'computer'
    ];

    Input::replace($input);

    Auth::shouldReceive('attempt')
          ->with($input)
          ->once()
          ->andReturn(true);

    $this->call('POST', 'login', $input);

    $this->assertRedirectedToRoute('dashboard');


}
```

Example of automated test using PHPUnit Testing Framework

Running of automated test using PHPUnit testing Framework

```
C:\xampp\htdocs\islington>phpunit
PHPUnit 4.5.1 by Sebastian Bergmann and contributors.

Configuration read from C:\xampp\htdocs\islington\phpunit.xml

.I....

Time: 1.07 seconds, Memory: 14.75Mb

OK, but incomplete, skipped, or risky tests!
Tests: 6, Assertions: 11, Incomplete: 1.
```

Product	Selenium	Katalon Studio	Unified Functional Testing	TestComplete	SoapUI
Available since	2004	2015	1998	1999	2005
Application Under Test	Web apps	Web (UI & API), Mobile apps	Web (UI &API), Mobile, Desktop, Packaged apps	Web (UI &API), Mobile, Desktop apps	API/Web service
Pricing	Free	Free	$$$$	$$$	$$
Supported Platforms	Windows Linux OS X	Windows Linux OS X	Windows	Windows	Windows Linux OS X
Scripting languages	Java, C#, Perl, Python, JavaScript, Ruby, PHP	Java/Groovy	VBScript	JavaScript, Python, VBScript, JScript, Delphi, C++ and C#	Groovy
Programming skills	Advanced skills needed to integrate various tools	Not required. Recommended for advanced test scripts	Not required. Recommended for advanced test scripts	Not required. Recommended for advanced test scripts	Not required. Recommended for advanced test scripts
Ease of Installation and Use	Require advanced skills to install and use	Easy to setup and use	Complex in installation. Need training to properly use the tool	Easy to setup. Need training to properly use the tool	Easy to setup and use

1.0 Maintenance

Maintenance activities include changing programs, procedures, or documentation to ensure

1. correct system performance;
2. Adapting the system to changing requirements;
3. Making the system operate more efficiently.

System Support

1.1 Maintenance - Types

Corrective

- Fixes errors and problems
- **Example**: Debugging program code

Adaptive

- Provides enhancements to a system
- **Example**: Add online capability, add features

Perfective

- Improves a system's efficiency, reliability, or maintainability
- **Example**: Optimizing the query, optimizing page load speed

Preventive

- Avoids future problems.
- Example: creating backup database, creating system restore point, creating history data

1.2 Maintenance - Occurrence

	Immediately After Implementation	Early Operational Life	Middle Operational Life	Later Operational Life
Corrective Maintenance	High	Low	Low	High
Adaptive Maintenance (Minor Enhancements)	None	Medium	Medium	Medium
Adaptive Maintenance (Major Enhancements)	None	None	Medium to High	Medium to High
Perfective Maintenance	Low	Low to Medium	Medium	Low
Preventive Maintenance	Low	Medium	Medium	Low