UECS3383 Software Quality Assurance

Lecture 06 – Testing, Review and Inspection

Definition of Software Testing

- Software testing is a **formal** process carried out by a **specialized testing team** in which a software unit, several integrated software units or an entire software package are examined by **running the programs on a computer**. All the associated tests are performed according to **approved test procedures** on **approved test cases**.

Software Testing - Objectives

- **Direct objectives**
  - To identify and reveal as many errors as possible in the tested software
  - To bring the tested software, after correction of the identified errors and retesting, to an acceptable level of quality.
  - To perform the required tests efficiently and effectively, within the limits budgetary and scheduling limitation.
- **Indirect objectives**
  - To compile a record of software errors for use in error prevention (by corrective and preventive actions)

Software Testing - Strategies

- Incremental Testing strategies:
  - Bottom-up testing (rule: bottom – up – left – right)
  - Top-down testing (rule: top – down – left – right)
- Big Bang Testing

Bottom-Up Testing

Top-Down Testing

Use of Stubs and Drivers – Incremental Testing

Top-down testing of module M8

Bottom-up testing of module M8

IEEE Definitions : Black Box Testing and White Box

- **Black box testing**
  - Testing that ignores the internal mechanism of the system or component and focuses solely on the outputs in response to selected inputs and execution conditions
  - Testing conducted to evaluate the compliance of a system or component with specified functional requirements
- **White box testing**
  - Testing that takes into account the internal mechanism of a system or component

White Box Testing – “Path” vs “Line” Coverage

- **Path coverage**
  - Path coverage of a test is measured by the percentage of all possible program paths included in planned testing.
- **Line coverage**
  - Line coverage of a test is measured by the percentage of program code lines included in planned testing.

White Box Testing

- **Advantages:**
  - Direct determination of software correctness as expressed in the processing paths, including algorithms.
  - Allows performance of line coverage follow up.
  - Ascertains quality of coding work and its adherence to coding standards.
- **Disadvantages :**
  - The vast resources utilized, much above those required for black box testing of the same software package.
  - The inability to test software performance in terms of availability (response time), reliability, load durability, etc.

Black Box Testing

- **Advantages:**
  - Allows us to carry out the majority of testing classes, most of which can be implemented solely by black box tests, i.e. load tests and availability tests.
  - For testing classes that can be carried out by both white and black box tests, black box testing requires fewer resources.
- **Disadvantages:**
  - Possibility that coincidental aggregation of several errors will produce the correct response for a test case, and prevent error detection.
  - Absence of control of line coverage. There is no easy way to specify the parameters of the test cases required to improve coverage.
  - Impossibility of testing the quality of coding and its strict adherence to the coding standards.

Black Box Testing – Equivalence Class (EC) Partitioning

- A black box method aimed at increasing the efficiency of testing and, at the same time, improving coverage of potential error conditions.
- An equivalence class (EC) is a set of input variable values that produce the same output results or that are processed identically.
- EC boundaries are defined by a single numeric or alphabetic value, a group of numeric or alphabetic values, a range of values, and so on.
- An EC that contains only valid states is defined as a "valid EC," whereas an EC that contains only invalid states is defined as the "invalid EC."
- In cases where a program's input is provided by several variables, valid and invalid ECs should be defined for each variable.

Black Box Testing – Equivalence Class (EC) Partitioning

- **According to the Equivalence Class Partition Method:**
  - Each valid EC and each invalid EC are included in at least one test case.
  - Definition of test cases is done separately for the valid and invalid ECs.
  - In defining a test case for the valid ECs, we try to cover as many as possible "new" ECs in that same test case.
  - In defining invalid ECs, we must assign one test case to each "new" invalid EC, as a test case that includes more than one invalid EC may not allow the tester to distinguish between the program's separate reactions to each of the invalid ECs.
  - Test cases are added as long as there are uncovered ECs.

Example: Entrance Ticket Price Table – The Pool

Entrance Ticket Price Table								
	Mon, Tue, Wed, Thurs, Fro				**Sat, Sun**			
Visitor's Status	OT	OT	M	M	OT	OT	M	M
Entry Hour	6:00 – 19:00	10:01 – 24:00	6:00 – 19:00	10:01 – 24:00	6:00 – 19:00	10:01 – 24:00	6:00 – 19:00	10:01 – 24:00
	Ticket Price ($)							
Visitor's Age								
0 – 16.00	5.00	6.00	2.5·	3.00	7.50	9.00	3.50	4.00
16.01 – 60.00	10.00	12.00	5.00	6.00	15.00	18.00	7.00	8.00
60.01 – 120.00	8.00	8.00	4.00	4.00	12.00	12.00	5.50	5.50

OT – One Time Customer
M – Member

Test Cases – The Ticket Price Module

Test case type	Test case no.	Day of week	Visitor's status	Entry hour	Visitor's age	Test case result
Valid ECs	**1**	**Mon.**	**Ot**	**7.55**	**8.4**	**$5**
	2	**Sat.**	**Mem**	**20.44**	**42.7**	**$8**
	3	**Sat.**	**Mem**	**22.44**	**65.0**	**$5.50**
	4	**Sat.**	**Mem**	**6.00**	**0.0**	**$3.50**
	5	**Sat.**	**Mem**	**19.00**	**16.0**	**$3.50**
	6	**Sat.**	**Mem**	**19.01**	**16.01**	**$8**
	7	**Sat.**	**Mem**	**19.01**	**60.0**	**$8**
	8	**Sat.**	**Mem**	**24.00**	**60.01**	**$5.50**
	9	**Sat.**	**Mem**	**24.00**	**120.0**	**$5.50**
Invalid ECs	**10**	**Mox.**	**Ot**	**7.55**	**8.4**	**Invalid day**
	11	**Mon.**	**88**	**7.55**	**8.4**	**Invalid visitor's status**
	12	**Mon.**	**Ot**	**4.40**	**8.4**	**Invalid entry hour**
	13	**Mon.**	**Ot**	**8@**	**8.4**	**Invalid entry hour**
	14	**Mon.**	**Ot**	**7.55**	**TTR**	**Invalid visitor's age**
	15	**Mon.**	**Ot**	**7.55**	**150.1**	**Invalid visitor's age**

Software Testing – Implementation

- **The Testing Process**
  - Determining the test methodology phase
  - Planning the tests
  - Test design
  - Test implementation
- **Test Case Design**
  - Test case data components
  - Test case sources
- **Automated Testing**
  - The process of automated testing
  - Types of automated testing
  - Advantages and disadvantages of automated testing
- **Alpha and beta site testing programs**

The Testing Process

Classification – Software Failure Damages to Customers and Users

- Endangers the safety of human beings
- Affects an essential organizational function with no system replacement capability available
- Affects functioning of firmware, causing malfunction of an entire system
- Affects an essential organizational function but a replacement is available
- Affects proper functioning of software packages for business applications
- Affects proper functioning of software packages for a private customer
- Affects functioning of a firmware application but without affecting the entire system.
- Inconveniences the user but does not prevent accomplishment of the system's capabilities

Classification – Software Failure Damages to Software Developers

- **Financial losses**
  - Damages paid for physical injuries
  - Damages paid to organizations for malfunctioning of software
  - Purchase cost reimbursed to customers
  - High maintenance expanses for repair of failed systems
- **Non-quantitative damages**
  - Expected to affect future sales
  - Substantially reduced current sales

Rating Units, Integrations and Applications

- The determination of priority in the testing plan are based on two factors:
  - **Factor A**: Damage Severity Level – the severity of results in case the module or application fails
  - **Factor B**: Software Risk Level – the level of risk represents the probability of failure. In order to determine the risk level of a module, unit, integration or application, the issues affecting risk require examination. These issues can be classified as module/application issues and programmer issues

Rating Units, Integrations and Applications

- Generating a combined rating based on the two factors:
  - The combined rating of "C" will be based on the rating for "A" (damage severity grading) and "B" (risk severity grading
  - **C = A + B; C = (k x A) + (m x B); C = A x B** **where k and m are constant**
- The inclusion of a unit, integration or application in a testing plan and the amount of resources allocated to each test depend of priorities as expressed in the combined rating.
- In order to determine these priorities, initial calculation of the ratings is necessary
- As a rule, the higher the rating, the higher the testing priority and the greater the allocation for testing resources

Issues Affecting Software Risk Level

- **Module/application issues**
  - Magnitude
  - Complexity and difficulty
  - Percentage of original software (vs. percentage of reused software)
- **Programmer issues**
  - Professional qualifications
  - Experience with the module's specific subject matter.
  - Availability of professional support (backup of knowledgeable and experience).
  - Acquaintance with the programmer and the ability to evaluate his/her capabilities.

Reasons for Changes in Testing Plan

- Changes may be required in the testing plan as a result of:
  - Unavailability of resources
  - Time requirements are too long and will cause the project to go beyond its completion schedule
  - Disagreements may arise about the evaluations of the expected damage and risk severity levels and/or about estimates of time and resources required for the testing activities
- The final testing plan will be completed only after these issues are resolved. The plan may need to be updated as the project proceeds, to reflect changes in conditions, including delays in project implementation

Example : Super Teacher

- "Super Teacher" is a software package designed to support teachers in managing the grades of elementary school pupils. The package includes eight applications. Ratings are required in order to plan the allocation of testing resources for each application.
- Applications 7 and 8 are based on high percentages of reused code. Application 2 was developed by team C, composed of new employees. A five-level scale is used for rating damage severity (Factor A) as well as software risk severity level (Factor B)
- The combined rating for the application is calculated according to three different methods:
- **C = A + B; C = (7 x A) + (2 x B); C = A x B**

Combined Rating Methods – Super Teacher Software

Application	Damage Severity (Factor A)	Damage Severity (Factor B)	A +B	7xA+2xB	A x B
1. Input of test results	3	2	5 (4-5)	25 (5)	6 (4)
2. Interface for input and output of pupils' data to and from other teachers	4	4	8 (1)	36 (1)	16 (1)
3. Preparation of lists of low achievers	2	2	4 (5-6)	18 (7)	4 (5-6)
4. Printing letters to parents of low achievers	1	2	3 (7-8)	11 (8)	2 (8)
5. Preparation of reports for the school principal	3	3	6 (3)	27 (3)	9 (3)
6. Display of a pupil's achievements profile	4	3	7 (2)	34 (2)	12 (2)
7. Printing of pupil's term report card	3	1	3 (7-8)	23 (6)	3 (7)
8. Printing of pupil's year-end report card	4	1	4 (5-6)	26 (4)	4 (5-6)

Note: Noteworthy that the two highest priorities were both determined in the same order by all three methods of calculating the combined rating.

* The figures in brackets in the combined rating columns represent the application preferences

Template: Software Test Plan (STP)

1 Scope of the tests

1.1 The software package to be tested (name, version and revision)

1.2 The documents that provide the basis for the planned tests

2 Testing environment

2.1 Sites

2.2 Required hardware and firmware configuration

2.3 Participating organizations

2.4 Manpower requirements

2.5 Preparation and training required of the test team

Template: Software Test Plan (STD)

3 Tests details (for each test)

3.1 Test identification
3.2 Test objective
3.3 Cross-reference to the relevant design document and the requirement document
3.4 Test class
3.5 Test level (unit, integration or system tests)
3.6 Test case requirements
3.7 Special requirements (e.g., measurements of response times, security requirements)
3.8 Data to be recorded

4 Test schedule (for each test or test group) including time estimates for:

4.1 Preparation
4.2 Testing
4.3 Error correction
4.4 Regression tests

Template: Software Test Description (STD)

1 Scope of the tests

1.1 The software package to be tested (name, version and revision)

1.2 The documents providing the basis for the designed tests (name and version for each document)

2 Test environment (for each test)

2.1 Test identification (the test details are documented in the STP)

2.2 Detailed description of the operating system and hardware configuration and the required switch settings for the tests

2.3 Instructions for software loading

3. Testing process

3.1 Instructions for input, detailing every step of the input process

3.2 Data to be recorded during the tests

4. Test cases (for each case)

4.1 Test case identification details

4.2 Input data and system settings

4.3 Expected intermediate results (if applicable)

4.4 Expected results (numerical, message, activation of equipment, etc.)

5. Actions to be taken in case of program failure/cessation

6. Procedures to be applied according to the test results summary

Implementation Phase Activities

Template : Software Test Report (STR)

1. Test identification, site, schedule and participation

1.1 The tested software identification (name, version and revision)
1.2 The documents providing the basis for the tests (name and version for each document)
1.3 Test site
1.4 Initiation and concluding times for each testing session
1.5 Test team members
1.6 Other participants
1.7 Hours invested in performing the tests

2. Test environment

2.1 Hardware and firmware configurations
2.2 Preparations and training prior to testing

Template : Software Test Report (STR)

3. Test results

3.1 Test identification

3.2 Test case results (for each test case individually)

4. Summary tables for total number of errors, their distribution and types

4.1 Summary of current tests

4.2 Comparison with previous results (for regression test summaries)

5. Special events and testers' proposals

5.1 Special events and unpredicted responses of the software during testing

5.2 Problems encountered during testing.

5.3 Proposals for changes in the test environment, including test preparations

5.4 Proposals for changes or corrections in test procedures and test case files

Test Cases – Types of Expected Results

- **Management information systems - expected results:**
  - Numerical
  - Alphabetic (name, address, etc.)
  - Error message. Standard output informing user about missing data, erroneous data, unmet conditions, etc.
- **Real-time software and firmware - expected results:**
  - Numerical and/or alphabetic massages displayed on a monitor's screen or on the equipment display.
  - Activation of equipment or initiation of a defined operation.
  - Activation of an operation, a siren, warning lamps and the like as a reaction to identified threatening conditions.
  - Error message. Standard output to inform the operator about missing data, erroneous data, etc.

Sources of Test Case

- **Random samples of real life cases** (Preferable – Stratified sampling of real life cases)
  - A sample of urban households (to test a new municipal tax information system)
  - A sample of shipping bills (to test new billing software)
  - A sample of control records (to test new software for control of manufacturing plant production)
  - A recorded of events that will be "run" as a test case (to test online applications for an internet site, and for real-time applications)

- **Synthetic test cases** (simulated test cases) prepared by test designers. This type of test case does not refer to an existing customer, shipment or product but to combinations of the system's operating conditions and parameters (defined by a set of input data)

Comparison of Test Data Sources

Implication	Random Samples of Cases	Synthetic Test Case
Effort required to prepare a test case file	Low effort, especially where expected results are available and need not be calculated	High effort; the parameters of each test case must be determined and expected results calculated
Required size of test case file	Relatively high as most cases refer to simple situations that repeat themselves frequently. In order to obtain a sufficient number of non-standard situation, a relatively large test case file needs to be compiled	Relatively small as it may be possible to avoid repetitions of any given combination of parameters
Efforts required to perform the software tests	High efforts (low efficiency) as tests must be carried out for large test case files. The low efficiency stems from the repetitive of case conditions, especially for the simple situations typical to most real-life case files	Low efforts (high efficiency) due to the relatively small test case file compiled so as to avoid repetitions

Comparison of Test Data Sources

Implication	Random Samples of Cases	Synthetic Test Case
Effectiveness – probability of error detection	• Relatively low – unless the test case files are very large due to the low percentage of uncommon combinations of parameters • No coverage of erroneous situations • Some ability to identify unexpected errors for unlisted situations	• Relatively high due to good coverage by design • Good coverage of erroneous situations by test case file design • Little possibility of identifying unexpected errors as all

Comparison of Test Data Sources

- In most cases, the test case file preferred should combined sample cases with synthetic cases so as to overcome the disadvantages of a single source of test cases and to increase the efficiency of the testing process
- In the case of combined test case files, test plans are often carried out in two stages:
  - First stage: synthetic test cases are used
  - Second stage: after correction of the detected errors, a random sample of test cases is used in the second stage

Test Cases for Reused Software

- It is quite common for reused software to include many application not required for the current software system in addition to the required applications
- In situation of this kind, planners should consider which reused software modules should be tested. Other modules of the reused software will not be tested

Peer Reviews

- Peer Reviews (inspections and walkthroughs) are directed at reviewing short documents, chapters or parts of a report, a coded printout of a software module
- Inspections and walkthroughs can take several forms and use many methods; usually, the reviewers are all peers, not superiors, who provide professional assistance to colleagues
- **Objective:** To detect as many design and programming faults as possible
- **Output:** A list of detected faults and for inspections, also a defect summary and statistics to be used as a database for reviewing and improving development methods

Objectives of Peer Reviews

- **Direct objectives**
  - To detect analysis and design errors as well as subjects where corrections, changes and completions are required
  - To identify new risks likely to affect the project.
  - To locate deviations from templates, style procedures and conventions.
  - To approve the analysis or design product. Approval allows the team to continue on to the next development phase.

- **Indirect objectives**
  - To provide an informal meeting place for exchange of professional knowledge about methods, tools and techniques.
  - To record analysis and design errors that will serve as a basis for future corrective actions.

Examples of Common Formal Design Reviews (DRs)

- DPR – Development Plan Review
- SRSR – Software Requirement Specification Review
- PDR – Preliminary Design Review
- DDR – Detailed Design Review
- DBDR – Data Base Design Review
- TPR – Test Plan Review
- STPR – Software Test Procedure Review
- VDR – Version Description Review
- OMR – Operator Manual Review
- SMR – Support Manual Review
- TRR – Test Readiness Review
- PRR – Product Release Review
- IPR – Installation Plan Review

Inspection vs Walkthrough

Participants of Peer Reviews

Inspection	Walkthrough
Review leader (moderator)	Review leader (coordinator)
The author	The author
Specialized professionals: • Designer • Coder or implementer • Tester	Specialized professionals: • Standards enforcer • Maintenance expert • User representative

Criteria: The Review Leader

- Be well versed in development of projects of the current type and familiar with its technologies. Preliminary acquaintance with the current project is not necessary
- Maintenance good relationships with the author and the development team
- Come from outside the project team
- Display proven experience in coordination and leadership of professional meetings
- For inspections, training as a moderator is also required

Specialized Professional – Inspections

- A **designer** – the system analyst responsible for analysis and design of the software system reviewed
- A **coder or implementer** – a professional who is thoroughly acquainted with coding tasks, preferably the leader of the designated coding team. This inspector is expected to contribute his or her expertise to the detection of defects that could lead to coding errors and subsequent software implementation difficulties
- A **tester** – an experienced professional, preferably the leader of the assigned testing team, who focuses on identification of design errors usually detected during the testing phase

Specialized Professional – Walkthroughs

- A **standard enforcer** – a team member who specializes in development standards and procedures, is assigned the task of locating deviations from those standards and procedures. Errors of this type substantially affect the team's long-term effectiveness, first because they cause extra difficulties for new members joining the development team, later because they will reduce the effectiveness of the team that will maintain the system

- A **maintenance expert** – is called upon to focus on maintainability, flexibility and testability issues, and to detect design defects capable of impeding correction of bugs or performance of future changes. Another area is documentation, whose completeness and corrections are vital for any maintenance activity

Specialized Professional – Walkthroughs

- A **standard enforcer** – participation of an internal (when the customer is a unit in the same firm) or an external user's representative in the walkthrough team contributes to the review's validity because he or she examines the software system from the point of view of the user-customer rather than the designer-supplier. In cases when a "real" user is not available, a team member may take on that role and focus on validity issues by comparing of the original requirements with the actual design

Process of Inspection vs Walkthrough

Inspection Process

- **Organizational Preparation**
- **Overview Meetings**
- **Thorough Review of Documents**
- **Inspection Session(s)**
  - Inspection session report
  - Inspection summary report
- **Corrections and Re-working**
- **Follow-up of Corrections and Re-working**

Walkthrough Process

- Organizational Preparation
- Brief Overview Reading
- Walkthrough sessions(s)
  - Walkthrough session report

Review Preparation

- The main tasks of the review leader in the preparation stage are:
  - To determine, together with the author, which sections of the design document are to be reviewed
    - Most difficult and complex sections
    - Most critical sections, where any defect can cause severe damage to the program application and thus to the user
    - Sections prone to defects
  - To select team members
  - To schedule peer review sessions – advise to limit it to two hours
  - To distribute the document to the team members prior to the review session

Sessions Documentation

- The documentation produced at the end of an inspection session is much more comprehensive than that of a walkthrough session

- Two documents are to be produced following an inspection session and subsequently distributed among the session participants:
  - **Inspection session findings report** – to assure full documentation of identified errors for corrections and follow-up
  - **Inspection session summary report** – summarizes the inspection findings and the resources invested in the inspection; it like-wise presents basic quality and efficiency metrics. It serves as input for analysis aimed at inspection process improvement and corrective action that go beyond the specific document or project

Post-Peer Review Activities

- The inspection process, contrary to the walkthrough, does not end with a review session or the distribution of report. Post-inspection activities are conducted to attest to:
  - The prompt, effective correction and reworking of all errors by the designer/author and his team, as performed by the inspection leader in the course of the assigned follow-up activities
  - Transmission of the inspection reports to the internal Corrective Action Board (CAB) for analysis. This action initiates the corrective and preventive actions that will reduce future defects and improve productivity

Peer Reviews – Sections Involved

Sections Recommended for Inclusion	Sections Recommended for Omission
Sections of complicated logic	"Straightforward" sections (no complication)
Critical sections, where defects severely damage essential system capability	Sections of a type already reviewed several times by the team in similar past projects
Sections dealing with new environments	Sections that, if faulty, are not expected to affect functionality
Sections designed by new or inexperienced team members	Reused design and code
NIL	Repeated parts of the design and code

Comparison of the Review Methodologies

Properties	Formal Design Reviews	Inspections	Walkthroughs
Main direct objectives	1. Detect errors 2. Identify new risks 3. Approve the design document	1. Detect errors 2. Identify deviations from standards	Detect errors
Main indirect objectives	Knowledge exchange	1. Knowledge exchange 2. Support corrective actions	Knowledge exchange
Review leader	Chief software engineer or senior staff member	Trained moderator (peer)	Coordinator (peer, project leader on occasion)
Participants	Top-level staff and customer representatives	Peers	Peers
Project leader participation	Yes	Yes	Yes, usually as the review's initiator
Specialized professionals in the team	_	1. Designer 2. Coder or implementer 3. Tester	1. Standard enforcer 2. Maintenance expert 3. User representative

Comparison of the Review Methodologies

Properties	Formal Design Reviews	Inspections	Walkthroughs
Process of Review:			
Overview meeting	No	Yes	Yes
Participant's preparations	Yes – thorough	Yes – thorough	Yes – brief
Review session	Yes	Yes	Yes
Follow-up of corrections	Yes	Yes	No
Infrastructure:			
Formal training of participants	No	Yes	No
Use of checklists	No	Yes	No
Error-related data collection	Not formally required	Formally required	Not formally required
Review documentation	Formal design review report	1. Inspection session findings report 2. Inspection session summary report	Walkthrough session findings report