02. Fundamentals of Testing

Causes of Software Defects

* Human error(mistake) produce
* Defects – faults/bugs in the program causes
* Failure - Fail to do what should or do something it shouldn’t

**Human Factor – human make mistakes:**

1. Poor Training
2. Time pressure
3. Code Complexity
4. Complexity of the infrastructure
5. Changing Technologies

**Organizational Factors**

1. Communication - Inefficient
2. Requirements – unclear defined

**Environmental conditions**

Radiation, magnetism, Electronic fields, pollution can change the hardware conditions

**Anomaly** – any conditions that deviates (отклонява) from expectations, based on requirements specifications, design documents, standards, etc. or someone’s perception or experience

**Error(Mistake)** – a human action that produce an incorrect result

**Bug/defect/fault/problem** – a flaw(недостатък) in a component or system that can cause them to fail to prform its required function

A defect, if encountered during execution, may cause a failure

**Failure** – Actual deviation(отклпнение) of the component or system from its expected delivery, service or result.

**Defect/fault masking** – an occurrence in which one defect prevents the detection of another

**Software quality** - The degree to which the component, system or process meets: specified requirements

customer or user needs or expectations

The Role of Testing

1. Reduce the risk of problems
2. Reduce long-term defect-related costs
3. Contribute to the quality of the software
4. Help meeting standards:

* Contractual or legal requirements
* Industry-specific standards
* Testing can give confidence in the quality of the software if it finds **few or no defects**
* If defects are found, the quality increases when those **defects are fixed**
* Lessons learnt from previous mistakes improve **future performance**

**How much testing should be done** is a matter of risk:

* Too much testing can delay the product release and increase the product price
* Insufficient testing hides risks of errors in the final product

What is Testing

* The process of **Exercising** software- to verify that it satisfied specific requirements and to detect errors
* The process of **Analyzing** a software item
* To detect differences between **existing and required conditions** (that is bug)
* To evaluate the **failures** of software item
* The process of **operating** a system or component under **specific conditions**
* Obverting or recording the **result**
* Making an **evaluation** of some aspect of the system component

**Main Testing Activities**

1. Planning and control
2. Choosing test conditions
3. Designing and executing test cases
4. Checking result
5. Evaluating exit criteria
6. Reporting on the testing process and system under test
7. Finalizing or completing closure activities

**Main Objectives in Testing**

1. Finding defects
2. Gaining confidence about the level of quality
3. Providing information for decision-making
4. Preventing defects

**Objectives of testing differ according the point of view**

* **Developers** - are percieved as very creative; rarely good communicators; specialize 1-2 skills (java, c++)
* **QA testers** – are perceived as very destructive; good communication skills, tact & diplomacy; needed to be multi-talented (technical testing, team skills)

**Debugging VS Testing**

* **Testing** - The activity that initially finds failures in a software item
* **Debugging** - The development activity that finds, analyses and removes the cause of the failure
* **Subsequent re-testing by a tester** - Ensures that the fix does indeed resolve the failure

Seven Tesying Principle

1. **Testing shows presence of defects**

* Testing can show that defects are present
* Cannot prove that there are no defects
* Appropriate testing reduces the probability for defects

1. **Exhostive (изчерпателен) testing is imposible**

* All combination of inputs and preconditions are usually infinite number
* Testing everything is not feasible (осъществимо)– except trivial cases
* Risk analysis and priorities should be used to focus testing efforts

1. **Early Testing**

* Start testing as early as possible
* Focused on defined objectives (цели)
* The later bug is found the more is cost

1. **Defect Clustering (Групиране на дефектите)**

* Testing effort shall be focused proportionally to the expected and later observed defect density of modules
* A small number of modules usually contains most of the defects discovered - responsible for most of the operational failures

1. **Pesticide Paradox**

* Same tests repeated over and over again tend to lose their effectiveness, because previously undetected defects remain undiscovered
* New and modified test cases should be developed

1. **Testing is context depend**

* Testing is done differently in different context

1. **Absence-of-errors fallacy (заблуда)–** finding and fixing defects itself does not help in these cases

* The system built is unusable
* Does not fulfill user needs and expectations

03. Fundamentals of Test Process

1. Fundamental Test Process

1.1 Test Planning and Control

1.2 Test Analysis and Design

1.3 Test Implementation and Execution

1.4 Evaluating Exit Criteria and Reporting

1.5 Test Closure Activities

2. Metrics and Measurement

3. The Psychology of Testing

Test Planning and Control

WHO: Test Menager, QA Leader or Senior QA

* Starts at the beginning of the software development project
* Must be regularly checked, updated, and adjusted

**Planning of recourses**

* Necessary resources:
* Which employees are needed, for what, when?
* How much time is needed?
* Which tools, equipment and utilities?
* Necessary training of the employees
* Organizational structure
* With the appropriate test management

**Test Control**

* **Monitoring** of test activities
* **Comparing** with the plan
* **Reporting** status of deviations from the plan
* Taking actionsfor **corrections**
* **Updating** the test plan according the feed back

**Test Prioritization**

* Software projects are often run under severe time pressure
* Prioritization guarantees that the critical software parts are tested first

**Test Plan**

* The results from the planning activities should be documented in a test plan
* The test plan is a formal document that describes how tests will be performed
* List of test activities to be performed to ensure meeting the requirements
* Features to be tested, testing approach, schedule, acceptance criteria

Test Analysis and Design

Can be considered as tow main tasks:

* Indentify test conditions
* Defining what should be tested
* An item or event of a component or system that could be verified by one or more test cases
* E.g., a function, transaction, feature, quality attribute, or structural element
* Designing test cases
* Defining what should be tested starts with reviewing the test basis
* Product specification may not be testable
* Unclear expected outcomes or behaviors
* Rework of the requirements has to be done

Test Basis

All documents from which the requirements of a component or system can be inferred (подразберат). The documentation on which the test cases are based. If a document can be amended only by way of formal amendment procedure (процедура за изменение), then the test basis is called a frozen test basis.

Test case

A set of input values, execution preconditions, expected results and execution postconditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance (съответствие) with a specific requirement

**Designing Test Cases**

According to the **level of concreteness** test cases can be logical and concrete

* Logical test cases – **defined first**, do not include concrete input/output values
* Concrete test cases – The **actual inputs** are chosen, **set priority**

*Initial situation* (precondition) must be described – **Environmental** conditions

*Which results and behavior are expected*

* Outputs
* Changes to global (persistent) data and states
* Any other consequences of the test case

*Test Cases can be designed for:*

* **Expected inputs**
* Specified behavior, output, and reaction
* Specified handling of exception and error cases
* **Unexpected inputs**
* Invalid and unexpected inputs or conditions
* Have no specified exception handling

The Test Oracle

* A mechanism for predicting the expected results – specification, similar product. The result can be inverted and compared to the initial input

Test Implementation and Execution

* Test conditions and logical test cases are transformed into concrete test cases
* The environment is set up to support the test execution activity
* Tests are executed and logged

**Test Case Execution**

* Follow the **priority** of the test cases set in the test plan
* Croup test cases in **test suits** – for fast execution and easier overview
* Start testing with **main functions** is recommended
* Failures occurred in that stage make further testing pointless – correction must be done before continuing
* **Time pressure** may cause running just a subset of all tests - Having test prioritized is important

**Test Protocol**

Tests without a protocol has no value

The test execution must be exactly and completely logged:

1. What tests were made
2. Who made the test
3. Which parts
4. When
5. How intensively
6. With what result
7. Software version

**Reproducibility**

* The tests must be easily repeated
* Test environment
* Input data
* Test logs, Etc.

.

**Failure Found**

1. Is it really a failure

* Erroneous(неправилен) or inexact test **specification**
* Problematic test **infrastructure** or test case
* Incorrect test **execution**

1. If it is a failure

* The failure must be **documented**
* Rough analysis of **possible causes**
* **Additional test cases** might be required

**Correction Might Lead to New Faults**

After each correction we must check:

* Is the fault **really corrected**
* Are there **new faults** introduced

Evaluating Exit Criteria and Reporting

* The set of generic and specific conditions for permitting a process of official completed, agreed with the stack holders
* Used to report against and to plan when to stop testing
* Were test exit criteria fulfill
* Test exit criteria might turn to be unrealistic - should be corrected

A simple example of test exit criteria might be:

100% statement coverage (each and every line of code needs to be checked and executed)

100% requirement coverage

all screens / dialogue boxes / error messages seen

100% of test cases have been run

100% of high severity faults fixed

80% of low & medium severity faults fixed

maximum of 50 known faults remain

maximum of 10 high severity faults predicted

time has run out

testing budget is used up

Test Summery Report

* Simple Message to the project manager – used in low level tests (component test)
* Formal Reports – for the stack holders – used in high level tests (integration, system)

Exit criteria at each level of testing - Component, Integrated, System, Acceptance

Test Closure Activities

Save the experience

The experience gathered should be analyzed and made available for future projects

* Achieved results
* Unexpected events - What were their causes?
* Open change requests - Why were they not implemented?
* User acceptance after deploying

Metrics and Measurement

Measurement – the concrete result from activity ex. 38 C temperature

Metrics – the conclusion from measurement for state – ex. illness

What can be subjected to a metric and tracked through measurement?

* **Test coverage**
* **Defects** - Including total found, total fixed, current backlog, average closure periods, and configuration, subsystem, priority, or severity distribution
* **Workload** and **resource** usage
* Planned and actual **costs**

Metrics and measurements should be applied throughout the software development lifecycle

* Should be aligned (приведен в съответствие) with project goals and objectives(задачи)
* This enables test analysts **to track and report test and quality results** to management in a consistent and coherent way each test level

A lack of metrics and measurements leads to purely **subjective assessments** of quality and testing

* This results in **disputes** over the meaning of test results toward the end of the lifecycle
* Also results in a lack of clearly perceived and communicated **value, effectiveness**, and efficiency for testing

The Psychology of Testing

* Different **mindset** is required:
* For testing and reviewing
* For developing software
* **Separation** of testing from development
* Helps focusing effort
* Avoids subjectivity

**Levels of independence - Tests can be designed by**:

* The person who wrote the software
* Another person (e.g. from another team)
* Person(s) from a different organizational group (independent test specialist)
* Person from a different organization or company

**Reporting Failures**

* Pointing out ones failures might be perceived as criticism
* Communicate bugs in a constructive way
* Start with **collaboration** rather than battles
* Focus on the **facts**, not the person
* Try to understand the way the other person **feels**
* Be sure the other person **understood** what you have said

04. Test Levels and Types

**Test Levels**

* Component Testing
* Integration Testing
* System Testing
* Acceptance Testing

**Test Types**

* Risk-Based Testing
* Functional Testing
* Non-functional Testing
* Structural Testing
* Testing Related to Changes: Re-testing and Regression Testing

**Maintenance Testing**

Component Testing

* Component testing - Testing separate components of the software
* Software units (components)
* Modules, units, programs, functions
* Classes – in Object Oriented Programming
* Respective tests are called - Module, unit, program or class tests

**Units vs. Components**

* **Unit** - The smallest compilable component
* Component
* A unit is a component
* The integration of **one** or more components is a component
* “**One”** stands for components that call themselves recursively

**Test Objects**

* Individual testing

Components are tested individually, Isolated from all other software components

* Isolation

Prevents **external influences** on the components

* Component test checks aspects internal **to the component**

Interaction with neighbors is not performed

**Component Testing Helpers**

* **Stubs**

In Component testing called components are replaced with stubs, simulators, or trusted components

* **Drivers**

Calling components are replaced with drivers or trusted super-components

Integration Testing

* Composing units to from larger structural units and subsystems
* Done by developers, testers or special integration team
* Supposes that components are already tested individually

**Levels of Integration Testing**

* Component integration testing
* Expose defects in the interfaces and interaction between integrated components
* Also call “integration test in small”
* System integration testing
* Testing the integration of system and packages
* Testing Interfaces to external organizations
* Also called “Integration test in large”

**Off-the-shelf Products – външни сървиси (плащане с карта)**

* Standard **existing components** used with some modification
* Usually not subject of component resting
* Must be tested for **integration**

**Why Integration Testing**

* After assembling the components **new fault may occur**
* Testing must confirm that all components **collaborate** correctly
* The main goal - **exposing faults**
* In the interfaces
* In the interaction between integrated components

**Some Typical Problems**

* Wrong **interface formats**
* Incompatible interface formats
* Wrong files format
* Typical faults in **data exchange**
* Syntactically wrong or no data
* Different interpretation of received data
* Timing problems

**Integration Approaches**

There are different approaches for integration testing

* The **Big Bang** approach -
* all components or systems are integrated simultaneously
* The main disadvantage: difficult to trace the cause of failures
* The **incremental** approach
* The main disadvantage: time-consuming

**Incremental Approaches**

* The **Top-Down** approach
* The high level logic and flow are tested first - the low level logic (components) are tested later
* The **Bottom-Up** approach
* Opposite to the Top-Down approach
* The main disadvantage - the high level or the most complex functionalities are tested late

System Testing

Why System Testing

* Previous tests were done against technical specifications
* The **system test**
* Looks at the system from another perspective
* Of the customer
* Of the future user
* Many functions and system characteristics result from the **interaction** of all system components

**Test Environment**

System testing requires specific test environment

* Hardware
* System software
* Device driver software
* Networks
* External systems
* Etc

A common mistake is testing in the **customer’s operational environment**

* Failures may cause damage to the syste
* No control on the environment
* Parallel processes may influence
* The test can hardly be reproduced

**Различни среди за различните видове тестване**

**Common Problems**

* Unclear or missing system requirements
* Missing specification of the system’s correct behavior
* Missed decision
* Not review, unclear and approved requirements
* Project failure possible
* Realization might turn to be in the wrong directions

Acceptance Testing

*Involving the Customer Himself*

**The Main Idea**

* The focus is on the customer's perspective and judgment
* Especially for customer specific software
* The customer is actually involved
* The only test he can understand
* Might have the main responsibility
* Performed in a customer’s like environment (UAT environment)
* As similar as possible to the target environment
* New issues may occur

**Forms of Acceptance Testing**

1. Contract fulfillment verification
2. User acceptance testing
3. Operational (acceptance) testing
4. Field test (alpha and beta testing)

**Contract Fulfillment Verification**

* Testing according to the contract
* Is the development / service contract accomplished
* Is the software free of (major) deficiencies (липси)
* Acceptance criteria
* Determined in the development contract
* Any regulations that must be adhered to
* Governmental, legal, or safety regulations

**Acceptance in Advance**

Acceptance test can be executed within a **lower test level**

* During **integration** – commercial off-the-shelf software
* During **component testing** – for component usability
* Before **system testing** – using a prototype and for a new functionality

**User Acceptance Testing**

* The client might not be the user
* Every user group must be involved
* Different user groups may have different expectation
* Rejection even by a single user group may be problematic

**Operational (Acceptance) Testing**

Acceptance by the system administrator

* Testing backup/restore cycles
* Disaster recovery
* User management
* Maintenance tasks
* Security Suitability

**Field Testing**

* Software may be run on many environments
* All variations cannot be represented in a test
* Testing with representative customers Alpha testing and Beta testing

Risk-Based Testing

**Risk**

* The possibility of negative or undesirable outcome event
* Any problem that may occur – would decrease perception (усещане) of product quality or project success

**Types of Risk**

* Product (quality) risk

The primary effect of a potential problem is on the **product quality**

* Project (planning) risk

The primary effect is on the **project success**

**Level of Risk**

* Not all risks are equal in importance
* Factors for classifying the level of risk:

1. **Likelihood (вероятност )**of the problem occurring

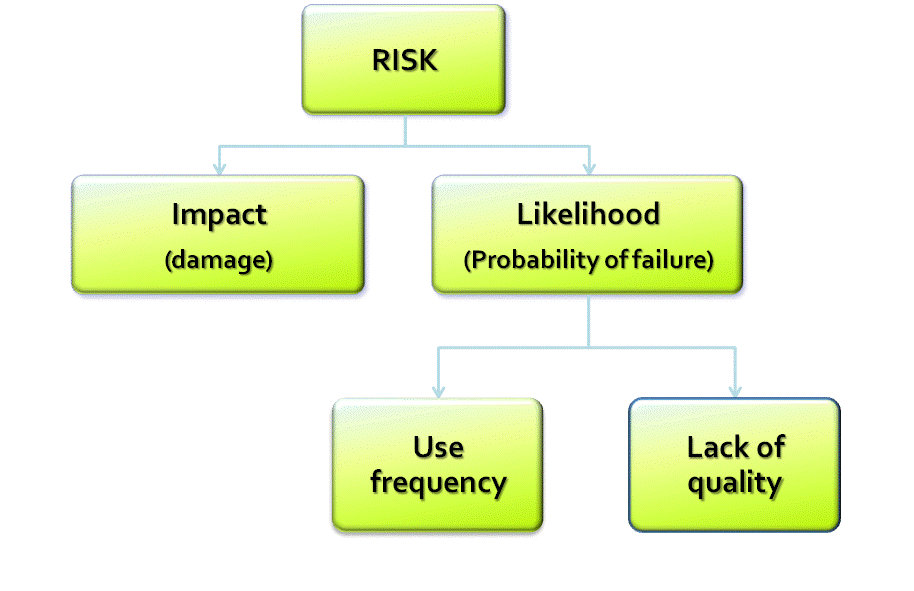
Arises from **technical considerations**

E.g. programming languages used, bandwidth of connections, etc.

2. **Impact (влияние, краен ефект))** of the problem in case it occurs

Arises from **business considerations**

E.g. financial loss, number of users affected, etc.



**Prioritization of Effort** - The more inportat risks are tested first

*Вероянността (Likelihood) може да се дъжи на честота на използване или липса на качество*

**Product Risks: What to Think About**

* Which functions and attributes (характеристики) are **critical** for the success of the product
* How **visible** is a problem in a function or attribute – for customers, users
* How often is the function **used**
* Can we do **without**

Functional Testing

* **Functional testing** verifies the system **input-output behavior**
* **Black box testing** method are used
* The test bases are functional requirements

**Functional Requirements (FR)**

They specify the **behavior** of the system

* **“What”** the system must be able to do
* Define **constrains** on the system

**Requirement Specifications**

FR must be documented

* Requirements Management System
* Text based Software Requirement Specifications **SRS**

**Requirements-based Testing**

* Requirements are used as the **basis for testing**

At least one test case for each requirement

Usually more than one is needed

* Mainly used in:

System testing

Acceptance testing

Structural Testing

*Testing the Software Structure/Architecture*

Examining the structure

* Offer referred to as “white-box” or ‘glass-box’ testing
* Use the information about the internal code structure or architecture
* Tools can be used to measure the code coverage of elements, such as statements or decision

Structure Testing Application

Mostly used for:

* Component Testing
* Integration Testing

Can also be applied at

* System integration
* Acceptance testing

Non-functional Testing

* “How well" or with what quality the system should carry out its function
* Attributive characteristics:
* Reliability
* Usability
* Efficiency

Testability of Requirements

* Nonfunctional requirements are often not clearly defined
* How would you test:

“The system should be easy to operate”

“The system should be fast”

* Requirements should be expressed in a testable way

Make sure every requirement is testable

Make it early in the development process

Nonfunctional Tests

1. Performance test - Processing speed and response time
2. Load test - Behavior in increasing system loads

* Number of simultaneous users
* Number of transactions

1. Stress test - Behavior when overloaded да се запазват данните word
2. Volume test - Behavior dependent on the amount of data
3. Testing of security

* Against unauthorized access
* Service attacks

1. Stability

* Mean time between failures колко често пада up and running
* Failure rate with a given user profile

1. Robustness test – как са handle exceptions

* Response
* Examination of exception handling and recovery to errors

1. Compatibility and data conversion

* Compatibility to given systems
* Import/export of data

1. Different configurations of the system - Back-to-back testing

Regression testing

1. Usability test

* Ease of learning the system
* Ease and efficiency of operation
* Understandability of the system

Testing Related to Changes:

Re-testing and Regression Testing

Re-Testing

* After a defect is detected and fixed the software should be retested, to confirm that original defect has been successfully removed
* This is called confirmation.

What is Regression Testing

* Retest of a previously tested program – after modigication
* Testing a newly introduced fault as a result of the changes made to the system
* May be performed at all test levels

Test Reusability

* Test cases used in regression testing run many times
* They have to be documented and reusable
* Strong candidates for test automation

Volume of the regression test

How extensive a regression test should be?

There are a few levels of testing extensity:

1. Defect retest (confirmation testing)-Rerunning tests that have detected faults
2. Testing altered functionality - Only changed or corrected parts
3. Testing new functionality - Testing newly integrated program parts
4. Complete regression test - Testing the whole system

Unexpected Side Effects

* The main trouble of software is code complexity
* Altered or new code parts may affect unchanged software
* Testing only code that is unchanged is not enough

Complete Regression Test

* The only way to be sure (as possible)
* **System environment** changes

Also require regression testing

Could have effects on every part of the system

* Too tim**e consuming** and **costly**

Not achievable in a reasonable cost

Impact analysis of the changes is needed

Maintenance Testing

Testing New Versions of The Software

Software does not wear out and tear

* Some design faults already exist
* Bugs are about to be revealed

A software project does not end with the first deployment

* Once installed, it will often be used for years or decades
* It will be changed, updated, and extended many times

New versions

* Each time a correction is made - a new version of the original product is created

Testing the changes can be difficult

* Outdated or missing system specifications

***Main Types of Maintenance***

* Adaptive maintenance - Product is adapted to new operational conditions
* Corrective maintenance - Defects being eliminated

**Common Reasons for Maintenance**

* The system is run under new operating conditions -

Not predictable and not planned

* The customers express new wishes
* Rarely occurring special cases
* Not anticipated (очакван) by design
* New methods and classes need to be written
* Rarely occurring crashes reported

**Testing After Maintenance**

* Anything new or changed should be tested
* Regression testing is required

The rest of the software should be tested for side effects

* What if the system is unchanged?

Testing is needed even if only the environment is changed

Software **sanity tests** are synonymous with **smoke** **tests**. A **sanity** or smoke test determines whether it is possible and reasonable to continue **testing**. It exercises the smallest subset of application functions needed to determine whether the systems are accessible and the application logic is responsive.

04a. Quality Attributes for Domain Testing

**Quality Attributes for Domain Testing**

* Functional Accuracy
* Functional Suitability
* Functional Interoperability (общи характеристики, качества, съвмвстимост)
* Accessibility
* Usability

**Testing of Software Characteristics**

We need to understand the main quality characteristics in order to:

* Recognize typical risks
* Develop appropriate testing strategies
* Specify effective tests

**Functional vs. Non Functional Testing**

* Functional testing - Focuses on **what** the system does
* Non functional testing - Focused on **how** the system does what it does

**Functional tests are based on functional requirements:**

* Written down requirements
* Implicit requirements
* Domain expertise of the tester

Functional Accuracy

* Does the System Give the Right Answers
* Concerned with adherence to specified or implied functional requirements
* Does the system give the right answer?
* Does the system produce the right effects?
* Also refers to the right degree of precision in the results

Computational Accuracy

Functional accuracy testing can include tests of computational accuracy

* Testing of computation accuracy is critical for applications with math-intensive functionality
* Applications for statistics, accounting, science, engineering, etc.

Functional Suitability

* Can the System Really Solve the Problem?
* Validation - Functional suitability is related to validation

Are we building the right system?

The capability of the software product to provide an appropriate set of functions for specified tasks and user objectives

Given the problem we need to solve; can the system solve it?

Functional Interoperability

* Do Separate Systems Speak the Same Language?

Дали дадена система се държи еднокво добре в различни среди

* Is an application able to function correctly in all intended target environments
* Hardware
* Software
* Connectivity infrastructure
* Database systems
* Operating systems

**Functional interoperability relates to:**

* Elements of the environment that the system must interoperate directly with
* Elements that it interoperates indirectly with
* Elements the system cohabitates with

**Cohabitation**

* Applications sharing common resources but without working together
* Network infrastructure
* CPU capability
* Memory space, Etc.

**What is a Good Interoperability?**

Good interoperability implies ease of integration with other systems

With few if any major changes

Accessibility

Tests the ability of users with particular requirements, restrictions, or disabilities to use the system

Accessibility of a system is usually required by different factors:

* National standards
* Voluntary guidelines
* Law or contract requirements

May force us to test every field and function with each assistive technology

Usability

**What is usability testing?**

Testing to determine the extent to which the software product is

* Understood
* Easy to learn
* Easy to operate
* Attractive to the users under specified conditions

Usability testing focuses on the users

**What Could Be Useful?**

Usability testing focuses on the users

Some knowledge in fields other than technology can be useful:

* Psychology
* Sociology
* Ergonomics
* Understanding of national standards related to accessibility

**Characteristics of a Usable Software**

* **Effectiveness**

Does the software enable the users to achieve their goals accurately and completely, under expected usage conditions

* **Efficiency**

Can users achieve his goals in some realistic, reasonable period

* **Satisfaction**
* **Understandability**

The simplicity or difficulty of figuring out:

* What the software does?
* Why you might need to use it?
* **Learnability**

The simplicity or difficulty of figuring out How to make the software do what it does?

* **Operability**

The degree of simplicity or difficulty inherent in carrying out certain distinct tasks within the software's feature set

* **Attractiveness**

The extent to which the software is visibly pleasing, friendly, and inviting to the use

**Usability Testing Techniques**

There are three main techniques for usability testing:

* Inspection (evaluation, review)
* Validation of the actual implementation
* Survey (questionnaire)

**Inspection (Evaluation, Review)**

* Involves considering the specification and designs from a usability point of view
* Effective and efficient way for early bug discovery

**Heuristic Evaluation**

Heuristic evaluation is a form of review

* A systematic inspection of a user interface design for usability
* It allows us to find usability problems in the design, resolve them, and then reevaluate - Cycle is repeated until wanted quality is achieved

**Validation of the Actual Implementation**

Running usability test scenarios

* Looking at usability attributes - E.g., speed of learning or operability
* May also include **pretest** and **posttest interviews** for the users
* Syntax tests - Evaluate what the interface allows / disallows
* Semantic tests - Evaluating meaningfulness of messages and outputs

**Surveys**

* Survey or questionnaire
* Another form of usability testing
* Can be used to gather observations of the users' behavior during interaction with the system in a usability test lab
* Standard and publicly available surveys
* Software Usability Measurement Inventory (SUMI)
* Website Analysis and Measurement Inventory (WAMMI)

06. Incident Menagment

Incident Management – Main Concepts

Testing often leads to observing **deviations from expected results**

Different names are used for that: Incidents, Bugs, Defects, Problems,Issues

**Incident vs. Bug – A Matter of Semantics**

Sometimes a distinction between incidents and bugs (defects) is made

* **Incident -** Any situation where the system exhibits questionable behavior
* **Bug -** An incident is referred to as a bug (defect) when the root cause is some problem in the item we're testing

**What Else Could Cause an Incident?**

* Misconfiguration or failure of the **test environment**
* Corrupted **test data**
* Bad **tests**
* Invalid **expected results**
* Tester **mistakes**

According to the test policy – any type of incident can be logged for tracking

**The Earlier – The Cheaper**

* Incident logging or defect reporting are not necessarily happening during testing
* Incidents can be logged, reported, tracked, and managed during development and reviews

**What Do We Report Defects Against?**

* The code or the system itself
* Requirements
* Design specifications
* User and operator guides and tests

**Glossary**

**Defect (bug)**

A flaw in a component or system that can cause the component or system to fail

**Error**

A human action that produces an incorrect result

**Failure**

Deviation of the component or system from its expected delivery, service, or result

**Incident**

Any event occurring that requires investigation. Occurs anytime the **actual results** of a test and the **expected results** of that test **differ**

**Incident logging**

Recording the details of any incident that occurred (e.g., during testing)

**Root cause analysis**

An analysis technique aimed at identifying the root causes of defects

Incident Reporting

**Managing Defects**

Defects found can reach count that is hard to manage

* A process for handling defects from discovery to final resolution is needed
* Should include reporting, classifying, assigning and managing defects

**Central Database**

A central database for each project should be established

* All incidents and failures discovered during testing are registered and administered
* Developers, QAs and stakeholders have access

**What Goes in an Incident Report?**

An incident report usually includes:

* Summary
* Steps to reproduce - Including inputs given and outputs observed
* Isolation steps tried
* Impact of the problem
* Expected and actual behavior
* Date and time of the failure
* Phase of the project
* Test case that produced the incident
* Name of the tester
* Test environment
* References to external sources
* Specification documents
* Various work items
* Attachments - Videos and screenshots
* Any additional information about the configuration
* Root cause of the defect

Usually set by the programmer, when fixing the defect

* Status and history information
* Comments
* Final conclusions and recommendations
* Severity and priority of the defect
* Sometimes classified by testers
* Sometimes a bug triage committee is responsible for that

Determines also the risks, costs, opportunities and benefits associated with fixing or not fixing the defect

**Defect Severity**

1. **Blocking**

* Stops the user from using the feature as it is meant to be used
* No reasonable workaround

1. **Critical**

* Data corruption
* Easily and repeatably throws an exception
* No reasonable workaround
* Feature does not work as expected

1. **High**

* Throws an exception when not following the happy path
* Confusing UI
* Has a reasonable workaround

1. **Medium**

* Feature works off the happy path with minor issues
* Small UI issues
* One or more reasonable workarounds

1. **Low**

* Cosmetic issues
* Many workarounds
* Low visibility to users

**Defect Priority**

Indicates how quickly the particular problem should be corrected

1 – Immediate

2 – Next Release

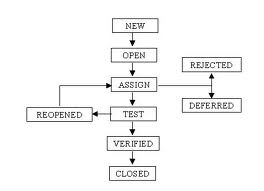
3 – On Occasion

4 – Open (not planned for now)

Defect Lifecycle

Фазите и етапита през които минава нашия бъг

Different defect-tracking systems may use different defect lifecycles



**Defect Lifecycle States**

1. **New**

* The bug is posted for the first time, the bug is not yet approved

1. **Open**

* The test lead approves that the bug is genuine, Changes the state as “OPEN”.

1. **Assign**

* The bug is assigned to corresponding developer or developer team

1. **Test**

* The bug has been fixed and is released to testing team

1. **Rejected**

* If the developer feels that the bug is not genuine, he rejects the bug

1. **Duplicate**

* The bug is **repeated twice** or two bugs mention the **same concept** of the bug

1. **Deferred**

The bug is expected to be fixed in next releases

Reasons for changing the bug to this status may have many factors:

* Bug may be low
* Lack of time for the release
* The bug may not have major effect on the software

1. **Verified**

* Once the bug is fixed and the status is changed to “TEST”, the tester tests the bug.
* If the bug is not present in the software, he approves that the bug is fixed

1. **Reopened**

* The bug still exists even after the bug is fixed by the developer
* The bug traverses the life cycle once again

1. **Closed**

* The bug is fixed, tested and approved

Metrics and Incident Management

**Defect Management Metrics**

Various metrics can be used for defect management during a project

* Helps managing defect trends
* Helps determining readiness for release

1. Total number of bugs
2. Number of open (active) bugs/tasks
3. Number of resolved bugs/tasks
4. Bugs per category
5. Bug cluster analysis
6. Defect density analysis
7. Number of defects discovered on a time unit - week, testing iteration, etc.
8. Mean-time to fix a defect

* The time between reporting and fixing/closing the bug

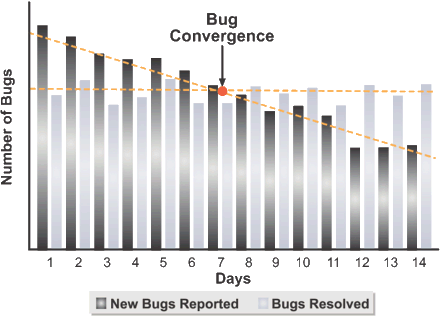
1. Time estimates versus actual time spent comparison

* Gives confidence in the estimates given by the team

**Bug Convergence**

**Фикснатите бъгове да со повече он новооткритите бъгове**

* Also called **open/closed charts**
* The point at which the rate of fixed bugs exceeds the rate of found bugs
* A visible indication that the team is making progress against the   
  active bug count
* A sign that the project end is within reach



**Defect Detection Percentage**

* Gives a measure of testing effectiveness
* Some defects are found prior to release while others - after deployment of the system
* **The defect detection percentage (DDP)** compares field defects with test defects, also called escaped defects

\_\_\_\_\_\_\_\_\_\_\_\_Defects testers\_\_\_\_\_\_\_\_\_

DDP = Defect testers + defects field

Some Golden Rules for Incident Reporting

**Golden Rules for Bug Reporting**

* **Watch your tests**
* Run your tests with care and attention
* You never know when you're going to find a problem
* **Reporting intermittent or sporadic symptoms**
* Some defects cannot be reproduced always
* Report how many times you **tried to reproduce** it and how many times it **did in fact occur**
* **Isolate the defect**
* Make carefully chosen changes to the steps used to reproduce it
* Move from boundary values to more generalized conditions
* **Provide information on the defect's impact**

Makes setting priority and severity easier and more accurate

* **Mind your language**
* Choose the right words in your report
* Be clear and unambiguous, neutral, fact-focused and impartial
* Be concise – avoid useless details
* **Make reviews of bug reports**

Make an experienced tester take a look at your report

07. Risk and Testing

**Risk and Testing – Main Concepts**

* Product Risks
* Project Risks
* Risk-Based Testing
* Risk Management
* Risk Identification
* Risk Analysis (Assessment)
* Risk Control

**Risk**

* The possibility of a negative or undesirable outcome or event
* Any problem that may occur would decrease perceptions of product quality or project success

**Two main types of risk are concerned**

* Product (quality) risks
* The primary effect of a potential problem is on the product quality
* Project (planning) risks
* The primary effect is on the project success
* Factors relating to the way the work is carried out

**Level of Risk**

Not all risks are equal in importance

Factors for classifying the level of risk:

* **Likelihood** of the problem occurring
* Arises from technical considerations
* E.g. programming languages used, bandwidth of connections, etc.
* **Impact** of the problem in case it occurs
* Arises from business considerations
* E.g. financial loss, number of users affected, etc

Product Risk

What is a product (quality risk) risk?

* The possibility that the system or software might fail to satisfy some reasonable customer, user, or stakeholder expectation
* Also referred to as "quality" risk

Example – missing feature

**Typical Product Risk**

What does "unsatisfactory software" mean?

* Omitted key functionality
* Unreliable and frequently fail to behave normally
* Might cause financial or other damage to users
* Poor software characteristics

Low security, usability, maintainability or performance

* Poor data integrity and quality

Project Risks

**Organizational factors:**

* Skill, training and staff shortages
* Complexity of the project team / organization
* Inadequate expectations or improper attitude toward testing

**Technical issues:**

* Ambiguous, conflicting or non-prioritized requirements
* Excessively large number of requirements
* High system complexity
* Quality problems with the design, the code or the tests
* Insufficient or unrealistic test environmentsE.g., not appreciating the value of testing

**Supplier issues:**

* Failure of a third party
* Contractual issues

Risk-Based Testing

**What is Risk-based testing?**

* An **approach** to testing that aims to:

Reduce the level of product risks

Inform stakeholders on their status

* Starts in the **initial stages** of a project
* Involves the **identification** of product risks and their use in guiding the test process

**Risk management** includes three primary activities:

* Risk identification
* Risk analysis
* Assessing the level of risk
* Risk control
* Mitigation
* Contingency
* Transference
* Acceptance

**Risk Identification Techniques (намиране на риска)**

Product and quality risks can be identified

* Expert interviews
* Project retrospectives
* Risk workshops and brainstorming
* Checklists
* Calling on past experience
* Include representatives of all (possible) stakeholders in risk identification

The broadest range of stakeholders will yield the most complete, accurate, and precise risk identification

Risk Analysis or Risk Assessment

**Risk analysis (assessment)** involves the **study of the identified risks**

* Categorize each risk item appropriately

Important for complex projects

* Assign each risk item an appropriate level of risk

Involves **likelihood** and **impact** as key factors

**Technical Factors for Assessing Likelihood**

* Complexity **of technology** and teams
* Personnel and training **issues**
* **Supplier and vendor** contractual problems
* Geographical distribution **of the development organization**

E.g., out-sourcing

* Legacy (established) versus new designs and technologies
* The quality (or lack of quality) in the tools and technology used
* Bad managerial or technical leadership
* Time, resource, and management pressure

Especially when financial penalties apply

* Lack of earlier testing and quality assurance tasks in the lifecycle
* High rates of requirements, design, and code changes in the project
* High defect rates
* Complex interfacing and integration issues

**Business Factors for Assessing Impact**

* Potential damage **to image**
* Loss of customers and **business**
* Potential financial, ecological, or **social losses or liability**
* Civil or criminal legal sanctions
* Loss of licenses, permits, etc.
* The lack of reasonable workarounds
* The visibility of failure and the associated negative publicity

**How Do We Determine the Level of Risk**

* Quantitatively
  + Using numerical ratings for both:
    - Likelihood (usually percentage)
    - Impact (often a monetary quantity)
    - Both can be calculated to a common risk index
* Qualitatively
  + E.g., very high, high, medium, low, very low

Risk Control

Mitigation (привенция)

* Taking preventive measures to reduce the likelihood and/or the impact of a risk

Contingency (План Б)

* Where we have a plan or perhaps multiple plans to reduce the impact if a risk should it occur

Transference (Застраховане на developer)

* Getting another party to accept the consequences of a risk should it occur

Accepting (ignoring) the risk

* A final option

**Techniques for Risk Control**

* Choosing an appropriate test design technique
* Reviews and inspection
* Reviews of test design
* Setting appropriate levels of independence

For the various levels of testing

* Using the most experienced person on test tasks
* Using strategies for confirmation testing (retesting) and regression testing

09. Web Testing

Web Testing Issues

* **Testing a web application** is quite a bit more difficult than testing the same functionality in a Windows Desktop application
* Various issues are present:
  + **Timing** difficulties
  + **Differences** between **browsers**
  + New technologies that make web browsing a rich, but **difficult-to-test**, experience
* Web browsers don't provide **clear visibility to what's happening on the page**
  + The tester has no straight-forward way of consistently **identifying an** **element** on a web page and **detecting changes** in the element's state
* New technologies introduce **new testing issues**
  + The same features that make web browsing a **richer experience** also **present obstacles to testing**
    - Asynchronous processes
    - Client side code running directly in the browser
    - Animation

**JavaScript**

* Before JavaScript the browser experience was completely **server based**
  + Entire pages were refreshed each time a button was clicked
* With JavaScript you can have an action take place instantly **without refreshing the page**
* This presents a whole **new set of paths** 
  + They need to be checked to **get full testing coverage**

**AJAX**

* AJAX = Asynchronous JavaScript and XML
  + JavaScript is capable of making calls to the server and updating selected portions of the page
    - From the testing perspective, AJAX may add a pause while information is retrieved from the server
* Handling AJAX
  + - One of the principal ways AJAX can be handled is by waiting for a particular element to reach some state -E.g. "text content = '1234'"

**Silverlight**

* Silverlight is a "plug in", object embedded to a standard web page
* Runs right in the browser
* Silverlight applications present unique testing issues
* E.g. the Silverlight elements are not readily accessible

Web Testing Tools

Web testing tools are a common type of test tool, used for:

* Functional/Regression Testing
* Load and Performance Testing
* Web Site Security Testing
* Web Site Management Testing
* Mobile Web/App Testing

**Web Testing Tools**

Web testing tools are a common type of test tool, used for:

* Scanning websites **for broken or missing hyperlinks** (Link checkers)
* Static analysis on HTML, checking for **conformance to standards**

Main Web Testing Methods

**Web Page Functionality Test (**Functionality Testing)

**1**) **Check all the links**

* the **outgoing** links from all the pages from specific domain under test
* all **internal** links
* links **jumping** on the same pages
* links **used to send the email**
* check if there are any **orphan** pages
* check for **broken** links

**2)Test forms in all pages**

* Check all the **validations** on each field
* Check for the **default values** of fields
* **Wrong inputs** to the fields in the forms
* Options to create forms if any, form delete, view or modify the forms

**3)** **Cookies testing**

* Test the application by **enabling** or **disabling** the cookies in your browser options
* Test if the cookies are **encrypted** before they are written to user machine
* Check application security by **deleting** the cookies

**4)Test HTML and CSS**

* Checking for **syntax errors**
* Readable **color schemas**
* Standard **compliance**

E.g. standards such W3C and ISO is followed

**5) Database testing**

Testing the backend databases, like **comparing the actual results with expected results.** Database testing basically include

* Data **validity** testing
  + - Test if any errors are shown while executing queries
* Data **Integrity** testing
  + - Maintained while creating, updating or deleting data in database
* Check **response time** of queries
* Test data retrieved from the database is shown accurately in the web application

**Web Page Usability Test** (Usability testing)

A method of **simulating the user's way of experience**

E.g., checking help links, contents in the page, checking menu options and their links, think times between the pages and message dialogs in the pages

**Test for navigation**

* How the user surfs the web pages, different controls like buttons, boxes or how user using the links on the pages to surf different pages
* Web site should be **easy to use**
* Instructions should be provided **clearly and correct** (it means whether they satisfy purpose)

**Content checking**

* Content should be **logical, easy** to understand and **meaningful**
* Check for **spelling errors**
* All the **anchor text** links should be working properly
* **Images** should be placed properly with proper sizes

**Interface Testing**

* The main layers are
* Web server
* Application server
* Database server
* Check if all the interactions between these servers are executed properly

**Compatibility Testing**

* Browser compatibility
* Operating system compatibility
* Mobile browsing
* Printing options

**Web Page Compatibility**

* A method of **testing multiple browsers** based on user requirements
* The web page presentation depends on **how well the components are used**

**Sample browser compatibility checklist**

|  |
| --- |
| CSS validation |
| HTML or XHTML validation |
| Page validations with and without JavaScript enabled |
| Ajax and jQuery functionality |
| Font size validation |
| Page layout in different resolutions |
| All images and alignment |
| Header and footer sections |
| Page styles |
| Date formats |
| Special characters with HTML character encoding |
| Page zoom-in and zoom-out functionality |

**Security Testing**

Security Testing basically include

* Testing for vulnerabilities of the web application
* Test **unauthorized access** to secure pages should not be permitted
* **Restricted files** should not be downloadable without appropriate access
* Check sessions are automatically killed after prolonged user inactivity

**Performance Testing**

* Verifies web page **responses as per expectations** based on the environment
* Also includes **stress testing** and **load testing** of the application
* Testing with Different Networks
* Required due to different points a user may access the system from
* E.g., **a local intranet** or an internet with a **lower network speed**
* An application's **performance** and the **accessibility** are based directly on the network used
* Using different operating systems and versions
* Using different databases

**Test Environment**

* Set up a test environment that is separate from your development and production environment
* This includes a separate web server, database server, and application server if applicable

Differences between Desktop, Client Server and Web Apps

1. Selenium

What is Selenium

* A Java Script based web testing tool
* Very popular Open Source Tool
* Support testing Web 2.0 Applications – on multiply OS and browsers
* Tests run directly in browser
* Implemented entirely using browsers technologies

(Изпълнено изцяло с помощта на технологии на браузъра)

Selenium’s Tool Suite

* Selenium IDE (Integrated Development Environment)
* Selenium 1 (Selenium RC or Remote Control) deprecated
* Selenium-Grid
* Selenium 2 (Selenium WebDriver)

**Areas of Application**

* Aacceptance/Functional testing
* Reproducing bugs
* Unit testing
* Regression testing
* Smoke testing

In computer programming and software testing, smoke testing (also confidence testing, sanity testing) is preliminary testing to reveal simple failures severe enough to reject a prospective software release.

Definition - What Does Smoke Testing mean?

Smoke testing, in the context of software development, is a series of test cases that are run before the commencement of more rigorous tests. The goal of smoke testing is to verify that an application's main features work properly. A smoke test suite can be automated or a combination of manual and automated testing.

Smoke testing is a preliminary test which is used to catch the high-level functional errors of an application. If the tests fail, then further testing of the application stops, and the build is refused for additional testing until the build passes the smoke test.

Selenium IDE

* Selenium IDE is a **prototyping tool** **for building test scripts**
* It is a **Firefox plugin**
* Provides an easy-to-use interface for **developing automated tests**
* Selenium IDE has a recording feature
* **Records user actions** as they are performed
* Exports them as a **reusable script** in a chosen programming language
* The script can be later **executed**
* Selenium IDE is simply intended as a **rapid prototyping tool**
* Not designed to **run test passes**
* Not designed to **build all the automated tests** you would need
* Doesn’t provide **iteration or conditional statements** for test scripts
* For serious test automation either **Selenium 2** should be used

With a supported programming language

**Building Test Cases**

* **Recording**
* **Adding Verifications and Asserts** With the Context Menu
* **Editing -** Editing commands and comments

Test cases can be grouped in test suites

**Recording Test Cases**

Selenium IDE automatically inserts commands into your test case based on your actions

* Clicking a link – **click** or **clickAndWait** commands
* Entering values – **type** command
* Clicking checkboxes or radio buttons – **click** command

**Base IRL**

The Selenium IDE allows pointing a base URL – allowing test cases to be run across different domains

**Selenium Commands “Selenese”**

* **Set of commands that run your tests**
* These commands essentially create a **testing language**
* A sequence of this commands is a **test script**

**What Can We Test?**

* **Existence of UI elements** based on their HTML tags
* Test for specific **content**
* Test for **broken links**
* **Input fields, selection list options, submitting forms**, and **table data**
* **Windows size, mouse position, alerts, Ajax functionality, popup windows, event handling, etc.**

**Scrip syntax**

* Command and 2 parameters
* The parameters are not always required – it depends on command
* Some cases required both, one or no parameters

Selenium parameters

Parameters vary, however they are typically:

* A locator for identifying a UI element within a page
* A text pattern for verifying or asserting expected page content
* A text pattern or a selenium variable for entering text in an input field or for selecting an option from an option list

Storing Selenium Scripts

Scripts for Selenium-IDE are stored in HTML text file format

This consists of an HTML table with three columns:

* Command
* Target
* Value

The second and third columns may not require values but they should be present

Test Suits

A test suite is a collection of tests

Test suites allow running the tests all at once, one after another as a one continuous batch-job

Selenium commands

A **command** is what tells Selenium **"what to do"**

Selenium commands come in **three “flavors”:**

* Actions
* Accessors
* Assertions

**Actions**

* Actions are commands that generally manipulate the state of the application
* They do things like:

“click this link” (click command)

“select that option” (select command)

* If an Action fails, or has an error, the execution of the current test is stopped

**The "AndWait" Actions**

Many Actions can be called with the “**AndWait”** suffix - E.g. “clickAndWait”

* When the browser makes a c**all to the server**
* Forces Selenium to **wait for a new page** to load
* Continuing to run commands before the page has loaded all its UI elements causes **unexpected test case failures**
* Do not use for actions that do not trigger a navigation/refresh

Accessors

* Accsessor examine the state of application and store the result in variable
* They are also used to automatically generate Assertions

Assertion

* Assertions are like Accessors, but they verify that the state of the application conforms to what is expected

E.g., “make sure the page title is X” (assertTitle) and “verify that this checkbox is checked” (verifyText)

Assertion Models

* **“assert”** When fails the test is aborted
* **“verify”** When fails, the test continues execution, logging the failure
* **”waitFor”** Waits for some condition to become true. Aborts the test when fails

**Assertion or Verification**

Choosing between assert and verify comes down to convenience and management of failures

* There is no point to check a paragraph if you are not on the correct page
* On the other hand, you may want to check many attributes of a page without aborting the test case
* Usually each command group is started with an assert followed one or more verify test commands
* Assert използваме за да сме сигурни че сме на точно определена страница, ако не сме няма смисъл да тестваме останалите неща

Some Common Selenium Commands

* open – opens a page or URL
* click/clickAndWait – performs a click operation, and optionally waits for new page to load – 30s by default (can be modify)
* verifyTitlle
* verifyTextPresent
* verifyElementPresent
* verifyText
* verifyTable
* waitForpageToLoad
* waitForElementPresent

Verifying Page Elements

Selenese allows multiply ways of checking for UI elements

* Is an element presence somewhere on the page
* Is a specific text – somewhere on the page
* Is a specific text at a specific location on the page

verifyTextPresent

Used to verify specific text exist somewhere on the page

It takes a single argument – the text pattern to be verify

verifyElementPresent

* Use this command when you must test for the presence of specific UI element, rather than its content
* This verification checks only tag
* One common use is to check for the presence of an image

verifyText

* verify both text and its UI element
* It must use a locator

Проверявя дали даден HTML element съдържа определен текст

Locating Elements

For many Selenium elements a target is required

*locatorType=location*

1. Locating by Identifier (id or name) *identifier=password*
2. Locating by id *id =loggingForm*
3. Locatig by name –

* locate first element
* if multiply elements, use filters
* default filter type is value

*name =continue value=Clear*

*name =continue Clear*

*name =continue type=button*

1. Locating by XPath

XPath is the language used for locating nodes in an XML (XHTML) document

* Useful when we don’t have a suitable id or name attribute for the element
* XPath locators can also be used to specify elements via attributes other than id and name
* Since only xpath locators start with “//”, the xpath= label can be omitted

xpath=//form[input/\@name=’username’] (4) - First form element with an input; child element with attribute named ‘name’ and the value ‘username’

1. Absolute vs. Relative Location

* Absolute
* Contains the location of all elements from the root (html) (E.g., xpath=/html/body/form)
* Very likely to fail after adjustments
* Relative
* Relative to an element that does have an id or name attribute
* E.g., //input[@name=’username’] - First input element with attribute named ‘name’ and the value ‘username

//form[1]

//form[@id=’loggingForm’]/input[4]

/html/body/div[5]/iframe

//\*[contains(@id,’logging’)]/input[4]

1. Locating Hyperlinks by Link Text

* Hyperlinks can be located in the webpage by using the text of the link
* If tow links with same text present, then the first match will be used

<a href= "continue.html">Continue</a>

*Link = Continue*

1. Locating by DOM

* Takes JavaScript that evaluates to an element on the page
* Can be simply the element’s location using the hierarchical dotted notation
* Since only dom locators start with “document”, the dom= label can be omitted

*Dom=document.forms[0].username*

*Document.forms[0].elements[3]*

*Document.forms[0].elements[continue]*

1. Locating by CSS

Use SCC selector

Faster than XPath

Can find the most complicated objects in an intrinsic HTML document

*1 <html>*

*2 <body>*

*3 <form id= "loginForm" >*

*4 <input class= "required" name= "username" type= "text"/>*

*5 <input class= "required passfield" name= "password"   
6 type= "password" />*

*7 <input name= "continue" type= "submit" value= "Login" />*

*8 <input name= "continue" type= "button" value= "Clear" />*

*9 </form>*

*10 </body>*

*11 <html>*

|  |  |
| --- | --- |
| **css=form#loginForm** | **3** |
| **css=input[name="username"]** | **4** |
| **css=input.required[type="text"]** | **4** |
| **css=#loginForm input:nth-child(2)** | **5** |

Matching Text Patterns

* Like locators, patterns are a type of parameter frequently required by Selenese commands
* E.g., verifyTextPresent, verifyTitle, verifyAlert, assertConfirmation, verifyText, and verifyPrompt
* Link locators can utilize a pattern
* Patterns allow describing what text is expected via the use of special characters

Globbing patterns

Selenium globbing patterns support only two special characters:

* "\*" (asterisk) - translates to “match anything,” i.e., nothing, a single character, or many characters
* [ ] (character class) - translates to “match any single character found inside the square brackets” – e.g., [aeiou]
* A dash (hyphen) can be used as a shorthand to specify a range of characters – e.g., [a-zA-Z0-9]
* Globbing patterns are prefixed with a glob: label
* Since globbing patterns are the default pattern – the label can be omitted

verifyTitle glob:Film\*Television\*

|  |  |  |
| --- | --- | --- |
| **Command** | **Target** | **Value** |
| **click** | **link=glob:Film\*Television Department** |  |
| **verifyTitle** | **glob:\*Film\*Television\*** |  |

* The actual link text on the page being tested can be

*“Film/Television Department”*

*“Film & Television Department”*

*“Film and Television Department”*

* The actual title of the page can be

“*De Anza Film And Television Department - Menu”*

*“Film & Television Department”*

Regular expressions patterns

* The most commonly used regular expression pattern is ".\*" ("dot star")

“0 or more occurrences of any character”

* Regular expression patterns in Selenese need to be prefixed with one of two possibilities:

regexp: (Case-sensitive)

regexpi: (Case insensitive)

Exact patterns

Exact patterns use no special characters

* They are used with exact: prefix
* Used when a special character need to be used as a literal
* Example: looking for an item labeled "Real \*"

select //selec glob:Real\* Matches anything or nothing after Real

select //selec exact:Real\* Matches the exact pattern

select //selec regexp:Real\\* Escaping is also possible

WaitFor Commands for AJAX Applications

* Using andWait commands will not work in AJAX driven web applications

Data is retrieved from server without refreshing the page

* We can use waitFor, waitForElementPresent or waitForVisible commands

They check for the desired condition every second and continue as soon as the condition is met

Ако JavaScript е изключен само CLICK

Store Commands and Selenium Variables

Selenium Variables can be used to store constants at the beginning of the script

Запаметявяме променливи когато се генерира автоматично някакво ID

* Can be used to store values passed to your test program from the command-line, from another program, or from a file
* Storing values with the store command

|  |  |  |
| --- | --- | --- |
| **Command** | **Target –** the text value to be stored | **Value** – the selenium variable |
| **store** | **paul@mysite.org** | **userName** |

Using the Stored Values

Accessing the store value is done by enclosing variable in curly brackets and preceded by a $ sign -$ {…}

|  |  |  |
| --- | --- | --- |
| **Command** | **Target** | **Value** |
| **verifyText** | **//div/p** | **${userName}** |
| **type** | **id=login** | **${userName}** |

Other Store Commands

* storeElementPresent
* Coresponds to verifyElementPresents
* stores a Boolean value true ot false, depends on whether the UI element is found
* storeText
* Corresponds to verifyText
* Uses a locater to identify specific page text - if found, it is stored in the variable
* StoreText can be used to extract text from the page being tested
* storeEval
* Takes a script as its first parameter
* Allows the test to store the result of running the script in a variable
* An equivalent store command exists for each verify and assert command

The eval() function evaluates JavaScript code represented as a string

JavaSctript Usage with Script Parameters

//ul[@id='lenta-butoni']/li[3]/

//div[@id='lenta-butoni-left']/ul/li[3]/a[@href='http://life.dir.bg']

//\*[@id='listView']/div[1]/dl/dd[2]

/html/body/div[3]/div[2]/div/div/div[1]/div[1]/dl/dd[2]/span[1]/span/input[2]

var d=new Date(); d.getDate()+'-'+((d.getMonth())) +'-'+d.getFullYear()

var d=new Date();((d.getMonth()))+' '+d.getDate()+' '+d.getFullYear()

' '

echo: 1-12-2015

December 01 2015, 00:00