SATHYABAMA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCSX1060 Software Testing UNIT IV –

TEST MANAGEMENT

DOCUMENTING TEST PLAN AND TEST CASE

TEST PLAN DEFINITION

A Software Test Plan is a document describing the testing scope and activities. It is the basis for formally testing any software/product in a project.

- **Test plan**: A document describing the scope, approach, resources and schedule of intended test activities. It identifies amongst others test items, the features to be tested, the testing tasks, who will do each task, degree of tester independence, the test environment, the test design techniques and entry and exit criteria to be used, and the rationale for their choice, and any risks requiring contingency planning. It is a record of the test planning process.
- **Master test plan**: A test plan that typically addresses multiple test levels.
- **Phase test plan**: A test plan that typically addresses one test phase.

TEST PLAN TYPES: One can have the following types of test plans:

- **Master Test Plan:** A single high-level test plan for a project/product that unifies all other test plans.
- **Testing Level Specific Test Plans:** Plans for each level of testing.
    - Unit Test Plan
    - Integration Test Plan
    - System Test Plan
    - Acceptance Test Plan
- **Testing Type Specific Test Plans:** Plans for major types of testing like Performance Test Plan and Security Test Plan.

TEST PLAN TEMPLATE

The format and content of a software test plan vary depending on the processes, standards, and test management tools being implemented. Nevertheless, the following format, which is based on IEEE standard for software test documentation, provides a summary of what a test plan can/should contain.

Test Plan Identifier:

- Provide a unique identifier for the document. (Adhere to the Configuration Management System if you have one.)

Introduction:

- Provide an overview of the test plan.
- Specify the goals/objectives.
- Specify any constraints.

References:

- List the related documents, with links to them if available, including the following:
    - Project Plan
    - Configuration Management Plan

Test Items:

- List the test items (software/products) and their versions.

Features to be tested:

- List the features of the software/product to be tested.
- Provide references to the Requirements and/or Design specifications of the features to be tested

Features Not to Be Tested:

- List the features of the software/product which will not be tested.
- Specify the reasons these features won't be tested.

Approach:

- Mention the overall approach to testing.
- Specify the testing levels [if it's a Master Test Plan], the testing types, and the testing methods [Manual/Automated; White Box/Black Box/Gray Box]

Item Pass/Fail Criteria:

- Specify the criteria that will be used to determine whether each test item (software/product) has passed or failed testing.

Suspension Criteria and Resumption Requirements:

- Specify criteria to be used to suspend the testing activity.
- Specify testing activities which must be redone when testing is resumed.

Test Deliverables:

- List test deliverables, and links to them if available, including the following:
    - Test Plan (this document itself)
    - Test Cases
    - Test Scripts
    - Defect/Enhancement Logs

- Test Reports

Test Environment:

- Specify the properties of test environment: hardware, software, network etc.
- List any testing or related tools.

Estimate:

- Provide a summary of test estimates (cost or effort) and/or provide a link to the detailed estimation.

Schedule:

- Provide a summary of the schedule, specifying key test milestones, and/or provide a link to the detailed schedule.

Staffing and Training Needs:

- Specify staffing needs by role and required skills.
- Identify training that is necessary to provide those skills, if not already acquired.

Responsibilities:

- List the responsibilities of each team/role/individual.

Risks:

- List the risks that have been identified.
- Specify the mitigation plan and the contingency plan for each risk.

Assumptions and Dependencies:

- List the assumptions that have been made during the preparation of this plan.
- List the dependencies.

Approvals:

- Specify the names and roles of all persons who must approve the plan.
- Provide space for signatures and dates. (If the document is to be printed.)

TEST PLAN GUIDELINES

- Make the plan concise. Avoid redundancy and superfluousness. If you think you do not need a section that has been mentioned in the template above, go ahead and delete that section in your test plan.
- Be specific. For example, when you specify an operating system as a property of a test environment, mention the OS Edition/Version as well, not just the OS Name.
- Make use of lists and tables wherever possible. Avoid lengthy paragraphs.

- Have the test plan reviewed a number of times prior to baselining it or sending it for approval. The quality of your test plan speaks volumes about the quality of the testing you or your team is going to perform.
- Update the plan as and when necessary. An out-dated and unused document stinks and is worse than not having the document in the first place.

The purpose of a test case is to describe how you intend to empirically verify that the software being developed conforms to the specifications. In other words, you need to be able to show that it can correctly carry out its intended functions. The test case should be written with enough clarity and detail that it could be given to an independent tester and have the tests properly carried out.

TEST CASE DESCRIPTION

A test case contains all the information necessary to verify some particular functionality of the software

Purpose: Describe the features of the software to be tested, and the particular behavior being verified by this test.

Requirement Traceability: A cross reference to the numbers of the requirements (in the system specification) which are being verified in this test.

Setup: Describe all the steps necessary to setup the software environment necessary to carry out the test.

Test Data:

- Write the actual input data to be provided and the expected output for your actual working product. You must provide the actual input data values, not just a description. For example, "Enter a new wholesale price" is wrong. "Enter a wholesale price of $23.50" is correct. For the expected results you must provide the actual values not just a description. For example, "The updated retail price is displayed" is wrong. "The retail price of $49.99 is displayed" is correct.

Be sure to include any manual calculations necessary to determine the expected outputs. (For example, if the program converts Fahrenheit temperature to Celsius, show the hand computations you did using the conversion formulas to arrive at the expected results).

- Often the test data can be shown in tabular form, with a column of input items and the corresponding column of expected outputs. If the test input is contained in an external data file, you can provide the file name and have the file contents listed on a separate page.

Example: This is a sample test case for the Pizza International application.

Test Case 1

Purpose: Verify that an alternative Locale can be selected from the combo box.
Requirement Traceability: Requirement number appears here.

Setup: Obtain the most current version of the Pizza International application. Follow the directions below.

Test Data:

Action	Input	Expected Output
Launch the application		The application main GUI appears. The Locale drop down box is displayed with "English" selected. Six toppings appear under the label "Toppings."
Verify all locales are available	Click the drop down button in the combo box	Verify that all four required locales appear: English, French, Spanish, Japanese
Select a language	Select "fr/French" in the combo box.	The topping names appear in French under the word "Garnitures"
Return to default	Select "en/English" in the combo box.	The original topping names are restored.

Note:

- We don't check for GUI colors or label positions that should be done in a GUI unit test.
- We don't check ALL locales that should be done in a unit test.
- We don't check the spelling of the toppings that should be done in a unit test.
- We don't check if the Submit button works, that should be done in a separate system test case.
  Optional: If you have a prototype GUI screen available, you can refer to it.

Important Software Test Metrics and Measurements – Explained with Examples and Graphs

In software projects, it is most important to measure the quality, cost and effectiveness of the project and the processes. Without measuring these, project can’t be completed successfully.

Software test metrics and measurements

There is a famous statement: *"We can't control things which we can't measure".* Here controlling the projects means, how a project manager/lead can identify the deviations from the test plan ASAP in order to react in the perfect time. Generation of test metrics based on the project needs is very much important to achieve the quality of the software being tested.

What are Software Testing Metrics?

A Metric is a quantitative measure of the degree to which a system, system component, or process possesses a given attribute.

Metrics can be defined as "STANDARDS *OF MEASUREMENT***".**

Software Metrics are used to measure the quality of the project. Simply, Metric is a unit used for describing an attribute. Metric is a scale for measurement.

Suppose, in general, "Kilogram" is a metric for measuring the attribute "Weight". Similarly, in software, "How many issues are found in thousand lines of code?", here No. of issues is one measurement & No. of lines of code is another measurement. Metric is defined from these two measurements.

Test metrics example:

- How many defects are existed within the module?
- How many test cases are executed per person?
- What is the Test coverage %?

What is Software Test Measurement?

Measurement is the quantitative indication of extent, amount, dimension, capacity, or size of some attribute of a product or process.

Test measurement example: Total number of defects.

Measure: Eg. Measuring distance in Km, m, cm etc.
Metrics: Eg. Km/hr, m/sec etc.

Why Test Metrics?

Generation of Software Test Metrics is the most important responsibility of the Software Test Lead/Manager.

Test Metrics are used to,

- Take the decision for next phase of activities such as, estimate the cost & schedule of future projects.
- Understand the kind of improvement required to success the project

Take decision on process or technology to be modified etc.

Importance of Software Testing Metrics:

As explained above, Test Metrics are the most important to measure the quality of the software.

Now, how can we measure the quality of the software by using Metrics?

Suppose, if a project does not have any metrics, then how the quality of the work done by a Test analyst will be measured?

For Example: A Test Analyst has to,

- Design the test cases for 5 requirements
- Execute the designed test cases
- Log the defects & need to fail the related test cases

After the defect is resolved, need to re-test the defect & re-execute the corresponding failed test case.

In above scenario, if metrics are not followed, then the work completed by the test analyst will be subjective i.e. the test report will not have the proper information to know the status of his work/project.

If Metrics are involved in the project, then the exact status of his/her work with proper numbers/data can be published.

I.e. in the Test report, we can publish:

1. How many test cases have been designed per requirement?
2. How many test cases are yet to design?
3. How many test cases are executed?
4. How many test cases are passed/failed/blocked?
5. How many test cases are not yet executed?
6. How many defects are identified & what is the severity of those defects?
7. How many test cases are failed due to one particular defect? etc.

Based on the project needs we can have more metrics than above mentioned list, to know the status of the project in detail.

Based on the above metrics, test lead/manager will get the understanding of the below mentioned key points.

a) % of work completed
b) % of work yet to be completed
c) Time to complete the remaining work
d) Whether the project is going as per the schedule or lagging? etc.

Based on the metrics, if the project is not going to complete as per the schedule, then the manager will raise the alarm to the client and other stake holders by providing the reasons for lagging to avoid the last minute surprises.

Metrics Life Cycle:

- **Analysis**
  - Identification of the metrics
  - Define the identified metrics
- **Communicate**
  - Explain the need of the metric to the stakeholder and the testing team.
  - Educate the testing team about the data points need to be captured for processing the metric.
- **Evaluation**
  - Capture and verify the data.
  - Calculating the metrics value using the data captured.
- **Report**
  - Develop the report with effective conclusion
  - Distribute the result to the stake holders and the respective representatives.
  - Take feedback from stake holders

Types of Manual Test Metrics:

Testing Metrics are mainly divided into 2 categories.

1. Base Metrics
2. Calculated Metrics

Base Metrics:

Base Metrics are the Metrics which are derived from the data gathered by the Test Analyst during the test case development and execution. This data will be tracked throughout the Test Life cycle. I.e. collecting the data like, Total no. of test cases developed for a project (or) no. of test cases need to be executed (or) no. of test cases passed/failed/blocked etc.

Calculated Metrics:

Calculated Metrics are derived from the data gathered in Base Metrics. These Metrics are generally tracked by the test lead/manager for Test Reporting purpose.

SOFTWARE ESTIMATION TECHNIQUES - COMMON TEST ESTIMATION TECHNIQUES USED IN SDLC

In order to successful software project & proper execution of task, the Estimation Techniques plays vital role in software development life cycle. The technique which is used to calculate the time required to accomplish a particular task is called *Estimation Techniques*. To estimate a task different effective **Software Estimation Techniques** can be used to get the better estimation.
Before moving forward let's ask some basic questions like What is use of this? or Why this is needed? or Who will do this? So in this article I am discussing all your queries regarding ESTIMATION.

Estimation

"Estimation is the process of finding an estimate, or approximation, which is a value that is usable for some purpose even if input data may be incomplete, uncertain, or unstable". The Estimate is prediction or a rough idea to determine how much effort would take to complete a defined task. Here the effort could be time or cost. An estimate is a forecast or prediction and approximate of what it would Cost. A rough idea how long a task would take to complete. An estimate is especially an approximate computation of the probable cost of a piece of work.

The calculation of **test estimation techniques** is based on:

- Past Data/Past experience
- Available documents/Knowledge
- Assumptions
- Calculated risks

Before starting one common question arises in the testers mind is that "Why do we estimate?" The answer to this question is pretty simple, it is to avoid the exceeding timescales and overshooting budgets for testing activities we estimate the task.

Few points need to be considered before estimating testing activities:

- Check if all requirements are finalize or not.
- If it not then how frequently they are going to be changed.
- All responsibilities and dependencies are clear.
- Check if required infrastructure is ready for testing or not.
- Check if before estimating task is all assumptions and risks are documented.

Software Estimation Techniques

There are different Software **Testing Estimation** Techniques which can be used for estimating a task.

- Delphi Technique
- Work Breakdown Structure (WBS)
- Three Point Estimation

- Functional Point Method

1) Delphi Technique:

This is one of the widely used software testing estimation technique. In the Delphi Method is based on surveys and basically collects the information from participants who are experts. In this estimation technique each task is assigned to each team member & over multiple rounds surveys are conduct unless & until a final estimation of task is not finalized. In each round the thought about task are gathered & feedback is provided. By using this method, you can get quantitative and qualitative results.

In overall techniques this technique gives good confidence in the estimation. This technique can be used with the combination of the other techniques.

2) Work Breakdown Structure (WBS):

A big project is made manageable by first breaking it down into individual components in a hierarchical structure, known as the Work breakdown structure, or the WBS. The WBS helps to project manager and the team to create the task scheduling, detailed cost estimation of the project. By using the WBS motions, the project manager and team will have a pretty good idea whether or not they've captured all the necessary tasks, based on the project requirements, which are going to need to happen to get the job done.

In this technique the complex project is divided into smaller pieces. The modules are divided into smaller sub-modules. Each sub-module is further divided into functionality. And each functionality can be divided into sub-functionalities. After breakdown the work all functionality should review to check whether each & every functionality is covered in the WBS.

Using this you can easily figure out the all task needs to completed & they are breakdown into details task so estimation to details task would be easier than estimating overall Complex project at one shot.

Work Breakdown Structure has four key benefits:

Work Breakdown Structure forces the team to create detailed steps -In The WBS all steps required to build or deliver the service are divided into detailed task by Project manager, Team and customer. It helps to raise the critical issues early on, narrow down the scope of the project and create a dialogue which will help make clear bring out assumptions, ambiguities, narrow the scope of the project, and raise critical issues early on.

Work Breakdown Structure help to improve the schedule and budget.
WBS enables you to make an effective schedule and good budget plans. As all tasks are already available so it helps in generating a meaningful schedule and makes scheming a reliable budget easier.

Work Breakdown Structure creates accountability **-** the level of details task breakdown helps to assign particular module task to individual, which makes easier to hold person accountable to complete the task. Also the detailed task in WBS, people cannot allow hiding under the "cover of broadness."

Work Breakdown Structure creation breeds commitment- The process of developing and completing a WBS breed excitement and commitment. Although the project manager will often develop the high-level WBS, he will seek the participation of his core team to flesh out the extreme detail of the WBS. This participation will spark involvement in the project.

3) Three Point Estimation:

Three point estimation is the estimation method is based on statistical data. It is very much similar to WBS technique, task is broken down into subtasks & three types of estimation are done on this sub pieces.

Optimistic Estimate (Best case scenario in which nothing goes wrong and all conditions are optimal.) = A

Most Likely Estimate (most likely duration and there may be some problem but most of the things will go right.) = M

Pessimistic Estimate (worst case scenario which everything goes wrong.) = B

Formula to find Value for Estimate (E) = A + (4*M) + B / 6

Standard Deviation (SD) = = (B – A)/6

Now a days, planning poker and Delphi estimates are most popular testing *test estimation techniques*.

4) Functional Point Method:

Functional Point is measured from a functional, or user, point of view.It is independent of computer language, capability, and technology or development methodology of the team. It is based on available documents like SRS, Design etc.

In this FP technique we have to give weightage to each functional point. Prior to start actual estimating tasks functional points are divided into three groups like Complex, Medium & Simple. Based on similar projects & Organization standards we have to define estimate per function points.

CONFIGURATION MANAGEMENT

The purpose of Software Configuration Management is to establish and maintain the integrity of the products of the software project throughout the project's software life cycle. Software Configuration Management involves identifying configuration items for the software project, controlling these

configuration items and changes to them, and recording and reporting status and change activity for these configuration items

Configuration management (CM) refers to a discipline for evaluating, coordinating, approving or disapproving, and implementing changes in artifacts that are used to construct and maintain software systems. An artifact may be a piece of hardware or software or documentation. CM enables the management of artifacts from the initial concept through design, implementation, testing, baselining, building, release, and maintenance.

At its heart, CM is intended to eliminate the confusion and error brought about by the existence of different versions of artifacts. Artifact change is a fact of life: plan for it or plan to be overwhelmed by it. Changes are made to correct errors, provide enhancements, or simply reflect the evolutionary refinement of product definition. CM is about keeping the inevitable change under control. Without a well-enforced CM process, different team members (possibly at different sites) can use different versions of artifacts unintentionally; individuals can create versions without the proper authority; and the wrong version of an artifact can be used inadvertently. Successful CM requires a well-defined and institutionalized set of policies and standards that clearly define

- the set of artifacts (configuration items) under the jurisdiction of CM
- how artifacts are named
- how artifacts enter and leave the controlled set
- how an artifact under CM is allowed to change
- how different versions of an artifact under CM are made available and under what conditions each one can be used
- how CM tools are used to enable and enforce CM
- These policies and standards are documented in a CM plan that informs everyone in the organization just how CM is carried out.
- Application to Core Asset Development

The entire core asset base is under CM, with support for all the tasks described in the preceding section. Core assets, after all, may be developed in parallel by distributed teams, may need their builds and releases to be managed, and so forth. Beyond this support, however, core asset development requires other features of the organization's CM capability. First, it requires a flexible concept of assets that encompasses hardware, software, and documents of all varieties. One of the more useful features of a CM tool is the ability to report the differences between two versions of an artifact. However, doing so often requires fluency in the language in which the artifact is represented. (If you've ever tried to execute a DIFF command on two binary files, you get the point.) Thus, a tool's difference-reporting capabilities may weigh heavily in the selection process.

Application to Product Development

The CM process should make it easy to set up the initial configuration of a new product. Each time a new product is developed (which can occur very often in a healthy and robust product line), the task of determining the appropriate core assets and how to make them available must be supported.

CM also has to keep track of all the versions of configuration items that are used, including the version of the tool environment (compilers, test suites, and so on) used to create the configuration. Incorporating new versions of core assets to build a new version of a product is a task that requires an impact analysis that must be supported by CM. Changes in core assets must be communicated via the CM process to the core asset development organization.

CMMI steps for CM: I Capability Maturity Model Integration, Version 1.1 for Systems Engineering and Software Engineering lists the following practices as instrumental for a CM capability in an organization:

- Identify the configuration items, components, and related work products that will be placed under configuration management.
- Establish and maintain a configuration management and change management system for controlling work products.
- Create or release baselines for internal use and for delivery to the customer.
- Track change requests for the configuration items.
- Control changes in the content of configuration items.
- Establish and maintain records describing configuration items.
- Perform configuration audits to maintain the integrity of the configuration baselines.

Practice Risks

CM imposes intellectual control over the otherwise unmanageable activities involved in updating and using a multitude of versions of a multitude of artifacts, both core assets (of all kinds) and product-specific resources. Without an adequate CM process in place, and without adequate, adherence to that process, developers will not be able to build and deploy products correctly, let alone recreate earlier versions of products. Inadequate CM control can result from the following:

- **an insufficiently robust process:** CM for product lines is more complex than CM for single systems. If an organization does not define a robust enough CM process, CM will fail, and the product line approach to product building will become less efficient.
- **CM occurring too late:** If the organization developing the product line does not have CM practices in place well before the first product is shipped, building new product versions or rebuilding shipped versions will be very time-consuming and expensive, negating one of the chief benefits of product lines.
- ***multiple core asset evolution paths*:** There is a risk that a core asset may evolve in different directions–something that can happen either (1) by design in order to enable the usage of a core asset in different environments such as multiple operating systems or (2) by accident when a core asset evolves within a specific product. When done by design, the evolution might be unavoidable and increase the complexity of the CM. You must watch for evolution by accident, because it can degrade the usefulness of the core asset base.

- **unenforced CM practices**: Owing to the complexity of the total product line configuration, not enforcing a CM process can result in total chaos (a result that's much worse than that for a single-system).
- **insufficiently robust tool support:** CM that is sophisticated enough to support a nontrivial product line requires tool support, and there is no shortage of available commercial CM systems. However, most of them do not directly support the functionality required for the CM to be useful in a product line context. Many of them can be "convinced" to provide the necessary functionality, but this convincing is a time-consuming task requiring specialized knowledge. If the organization fails to assign someone to customize the CM system for the product line, the CM tool support is likely to be ineffectual. Such a person needs to have both a good understanding of the product line processes and a solid grounding in CM.

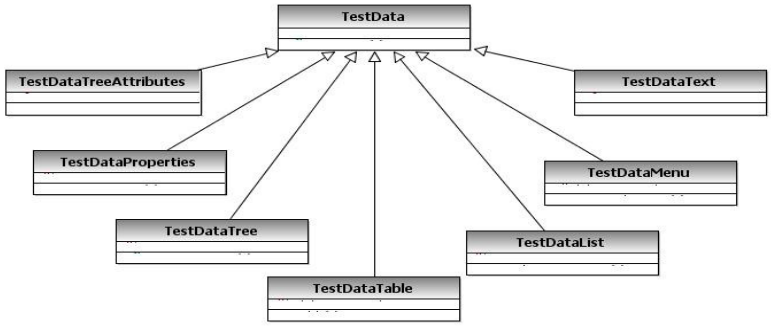
What is the ideal test data?

Test data can be said to be ideal if for the minimum size of data set all the application errors get identified. Try to prepare test data that will incorporate all application functionality, but not exceeding cost and time constraint for preparing test data and running tests.

How to prepare test data that will ensure complete test coverage?

Test data set examples:

1) No data: Run your test cases on blank or default data. See if proper error messages are generated.
2) Valid data set: Create it to check if application is functioning as per requirements and valid input data is properly saved in database or files.
3) Invalid data set: Prepare invalid data set to check application behavior for negative values, alphanumeric string inputs.
4) Illegal data format: Make one data set of illegal data format. System should not accept data in invalid or illegal format. Also check proper error messages are generated.
5) Boundary Condition data set: Data set containing out of range data. Identify application boundary cases and prepare data set that will cover lower as well as upper boundary conditions.
6) Data set for performance, load and stress testing: This data set should be large in volume.



class diagram shows the predefined test data types that Functional Tester makes available

TEST DATA MANAGEMENT

When test data plays such an important role in assuring the quality of the product, it's reasonable to say that its management and streamlining also plays an equally important role in Quality Assurance of any product that has to be released to the customers.

Need for test data management and best practices:

- A large number of organizations are having rapidly changing business goals to cater to the end user needs and hence it's needless to mention that the appropriate test data is instrumental in determining the quality of the testing. This will involve setting up the exact kind of data for the respective test environments and monitoring the behavioral patterns. As already discussed, a large chunk of a testing team's time is expended in planning of test data and its related tasks. Many a times testing of any functionality tends to be majorly hampered due to the non availability of appropriate test data which poses a critical challenge with respect to complete testing coverage.

- Also sometimes for certain testing requirements test data needs to be constantly refreshed. This itself causes a lot of delay in the cycle because of constant re-work which also increases the cost of the application reaching the market. In certain other times if the product being shipped has an involvement with different work group units in a large organization, the creation and refreshing of test data necessitates an intricate level of co-ordination across these work groups.

- Even though the test teams need to create all kinds of data that is possible to ensure adequate testing, organizations must also consider that doing this would mean that all the different kinds of data need to be stored in some kind of a repository. Although having a repository is good practice, storing excessive and unwanted data would not only significantly increase the storage space to store these large chunks of data but also make it increasingly challenging to fetch the appropriate data for the testing in question if there is no version maintenance and archiving of this repository.

Most of the organizations are generally faced with these common challenges with respect to test data. Thus, there needs to be some management strategies that need to be put into place to minimize the degree of these challenges.

Here below are some suggested methodologies for the management of the test data and keep it relevant to the testing needs. The following practices are very basic and generic which will commonly work for most organizations. How it is adopted, is purely the discretion of the respective organizations.

TEST DATA MANAGEMENT STRATEGIES

1. Analysis of data

Generally test data is constructed based on the test cases to be executed. For example in a System testing team, the end to end test scenario needs to be identified based on which the test data is designed. This could involve one or more applications to work. Say in a product which does workload management – it involves the management controller application, the middleware applications, the data base applications all to function in co-relation with one another. The required test data for the same could be scattered. A thorough analysis of all the different kinds of data that maybe required has to be made to ensure effective management.

2. Data setup to mirror the production environment

This is generally an extension from the previous step and enables to understand what the end user or production scenario will be and what data is required for the same. Use that data and compare that data with the data that currently exists in the current test environment. Based on this new data may need to be created or modified.

3. Determination of the test data clean-up

Based on the testing requirement in the current release cycle (where a release cycle can span over a long time), the test data may need to be altered or created as stated in the above point. This test data although not immediately relevant, maybe required at a later point. Hence a clear process of deeming when the test data can be cleaned up should be formulated.

4. Identify sensitive data and protect it

Many times in order to properly test applications, there may be large amount of very sensitive data that is required. For example, a cloud based test environment is a popular choice because it renders on demand testing of different products. However something as basic as guaranteeing user privacy in a cloud is cause of concern. So especially in cases where we will need to replicate the user environment, the mechanism to shield sensitive data must be identified. The mechanism is largely governed by volume of the test data used.

5. Automation

Just as we adopt automation for running repetitive tests or for running the same tests with different kinds of data, it's also possible to automate the creation of test data. This would help in exposing any errors that may occur with respect to data during testing. A possible way to do this is by comparing

the results that are produced by a set of data from consecutive test runs. Next automate this process of comparing.

6. Effective data refresh using a central repository

This is by far the most important methodologies and forms the heart of implementing data management. All of the points mentioned above, especially those with respect to data setup, data clean up are directly or indirectly co-relate with this.

A lot of effort in creating test data can be saved by maintaining a central repository which contains all kinds of data that maybe required for various kinds of testing. How is this done? In consecutive test cycles, for either a new test case or modified test case check if the data exists in the repository. If not existing, feed that data in the test environment first.

Next, this can be directed to this repository for future reference. Now for consecutive release cycles, the test team can use all or a subset of this data. Isn't the advantage very apparent? Depending on the sets of data that are frequently used, obsolete data can be easily eliminated and hence ensuring that correct data is always present, thereby reducing cost to store that unneeded data. Secondly, you can also have a couple versions of this repository saved or can revise it as necessary. Having different versions of the repository can help greatly in regression testing to identify what change in data can cause the code to break.

Test management tools

Test management tools can provide a lot of useful information, but the information as produced by the tool may not be in the form that will be most effective within your own context. Some additional work may be needed to produce interfaces to other tools or a spreadsheet in order to ensure that the information is communicated in the most effective way.

A report produced by a test management tool (either directly or indirectly through another tool or spreadsheet) may be a very useful report at the moment, but may not be useful in three or six months. It is important to monitor the information produced to ensure it is the most relevant now.

It is important to have a defined test process before test management tools are introduced. If the testing process is working well manually, then a test management tool can help to support the process and make it more efficient. If you adopt a test management tool when your own testing processes are immature, one option is to follow the standards and processes that are assumed by the way the tool works.

This can be helpful; but it is not necessary to follow the vendor-specific processes. The best approach is to define your own processes, taking into account the tool you will be using, and then adapt the tool to provide the greatest benefit to your organization.

Testopia for Test Management

Testopia is a test case management extension for Bugzilla. It is designed to be a generic tool for tracking test cases, allowing for testing organizations to integrate bug reporting with their test case run results. Though it is designed with software testing in mind, it can be used to track testing on virtually anything in the engineering process**.**

Features

1. Test case management extension for Bugzilla.
2. Tracks test cases, to integrate bug reporting with their test case run results.
3. Integrates with Bugzilla products, components, versions, and milestones to allow a single management interface for high level objects.
4. Allows users to login to one tool and uses Bugzilla group permissions to limit access to modifying test objects.
5. Testopia allows users to attach bugs to test case run results for centralized management of the software engineering process.
6. Requirements:
    - Bugzilla 3.6.x
    - Mysql 5.0 or PostGres 8.1.x
    - Mozilla compatible browser
    - Additional Perl Modules: Text::CSV XML::Schema Validator XML::Schema::Parser (for importer) and JSON 2.10

Testopia for Projects

- Projects can document their Integration (and Unit) tests in Testopia as well (under product project)
- Test Plans will be created for projects features
- Test Plans can be broken down into one or more Test Cases
- A Test Case is a collection of actions and conditions needed to verify a small feature or a sub-feature
- For debugging and clarity purposes, it is advisable to have many small Test Cases instead of one big Test Case
- Test Cases should run totally independent from each other. This will set the minimum for a Test Case

References

1. http://www.qatestingtools.com/Mozilla/Testopia
2. http://www.softwaretestingclub.com/forum/topics/test-case-management-tool