Experiment No. 01 – Problem Statement

Aim:
To develop a problem statement for applying various software engineering process activities

Theory:
1. Describe ideal/current state of affairs.
2. Explain your problem
3. Backup your assertions
4. Propose your solution
5. Explain the benefits of solution
6. Conclude by summarizing the problems and solution

Output:
Attach the pdf file for your problem statement.

Conclusion:
Insert your own conclusion for the experiment.

Experiment No. 02 - Traditional Process Model

Aim:
To apply traditional process model for the selected case study

Theory:
A software process model is defined as simplified process representation of software process each methods represent a process from a specific perspective.

Sample – Waterfall Model:
The waterfall model is also called as 'the linear-sequential model' or 'classic life' cycle model. The software development starts with requirements gathering phase. Then progress through analysis, design, coding, testing & maintenance.

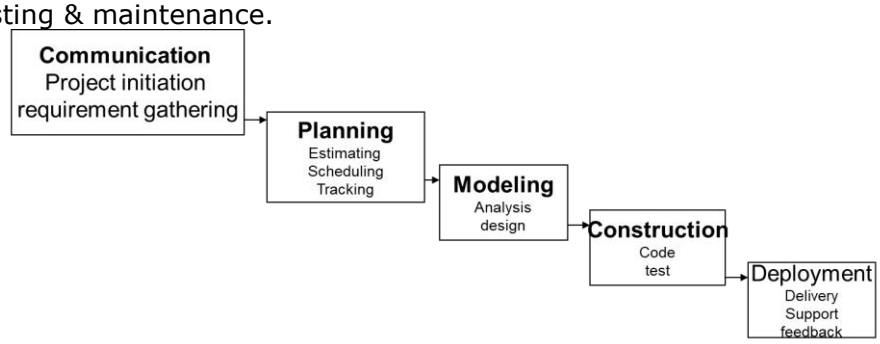


Fig.: Water fall Model

Advantages:
1. It is very simple to understand.
2. For implementation of small systems waterfall model is useful.

Disadvantages:
1. It is not useful for large projects.
2. It is very difficult to modify systems requirement if the middle of the development process.
3. It is not suitable for projects in which requirement software are not clear initially.

Suitability Justification:
We have used waterfall model for the projects because requirements of projects are very well known. Clear and fixed. Our project is very small to be implemented. In our project there is no need of customer involvement in the project. Development cycle product definition of project is not changed frequently. In our project there is no need of our participation in all phase. Hence the waterfall model is suitable process model for our project.

Procedure:

1. Explain the selected process model along with the diagram, pros and cons.
2. Mention the suitability reasons or justifications for selecting the process model.

Output:

Attach the pdf file for the selected traditional process model along with the justification.

Conclusion:

Insert your own conclusion for the experiment.

Experiment No. 03 – Agile Process Model

Aim:

To apply agile process model for the selected case study

Theory:

In earlier days Iterative Waterfall model was very popular to complete a project. But nowadays developers face various problems while using it to develop software. The main difficulties included handling change requests from customers during project development and the high cost and time required to incorporate these changes. To overcome these drawbacks of traditional models, in the mid-1990s the Agile Software Development model was proposed.

The Agile model was primarily designed to help a project to adapt to change requests quickly. So, the main aim of the agile model is to facilitate quick project completion. To accomplish this task agility is required. Agility is achieved by fitting the process to the project, removing activities that may not be essential for a specific project. Also, anything that is wastage of time and effort is avoided.

Actually Agile model refers to a group of development processes. These processes share some basic characteristics but do have certain subtle differences among themselves. In the Agile model, the requirements are decomposed into many small parts that can be incrementally developed. The Agile model adopts Iterative development. Each incremental part is developed over an iteration. Agile model is the combination of iterative and incremental process models.

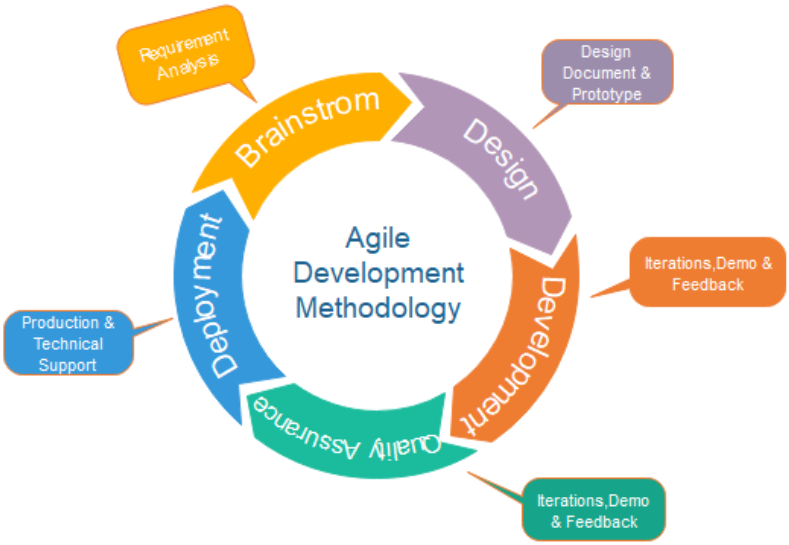


Fig.: Agile Model

Procedure:

1. Explain the Agile process model.
2. Mention the suitability reasons/justifications for selecting the agile process model.

Output:

Attach the pdf file for selected agile process model along with the justification.

Conclusion:

Insert your own conclusion for the experiment.

Experiment No. 04 – Use Case Diagram

Aim:
To implement use case diagram for the selected project

Theory:

Use case diagram represents the overall scenario of the system. A scenario is nothing but the sequence of steps describing an interaction between a user and a system. Thus the use case is a set of scenario tied together by some goal. To use case diagrams are drawn for exposing the view of the system. Use case diagrams contain following notations:-

1. Actor: An actor is an entity which interacts with the system. Actor carries out the use cases. A single actor may perform many use cases; a use case may have several actors. It is not necessary that the user should be an actor. The external system that gets some values or produces some value can be an actor. Actor is nothing but a role played by a person, system, device, or even an enterprise that has the state in the successful operation of the system.
2. Use cases: The use cases represent the behavior of the system. Typically functions are represented as the use cases. Use cases are represented in an oval shaped circle in which the title of function is given.
3. Association: It identifies an interaction between actors & use cases. Each association represents a dialog.
4. Include relationship: This relationship identifies a reasonable use case that is unconditionally required for execution of another use case. The decision about when and why to use the invented use case should be used is taken by extending use case itself.
5. Extend relationship: This identifies a reusable use case that conditionally interrupts the execution. Decision about when and why to use the invented use case should be used is taken by extending use case itself.
6. Generalization: This identifies an inheritance relationship between actors or between use cases.

Procedure:

1. Download and Install software tool like StarUML/Dia
2. List down system capabilities, actors and their relationships.
3. Develop use case diagram.

Output:
Attach the snapshot of use case diagram.

Conclusion:
Insert your own conclusion for the experiment.

Experiment No. 05 – Data Flow Diagram

Aim:
To perform structured data flow analysis using Data Flow Diagram.

Theory:

Data flow diagram is graphical representation of flow of data in an information system. It is capable of depicting incoming data flow, outgoing data flow and stored data. The DFD does not mention anything about how data flows through the system. There is a prominent difference between DFD and Flowchart. The flowchart depicts flow of control in program modules. DFDs depict flow of data in the system at various levels. DFD does not contain any control or branch elements.

Components of DFD:

External Entity: an outside system that sends or receives data, communicating with the system being diagrammed. They are the sources and destinations of information entering or leaving the system. They might be an outside organization or person, a computer system or a business system. They are also known as terminators, sources and sinks. They are typically drawn on the edges of the diagram.

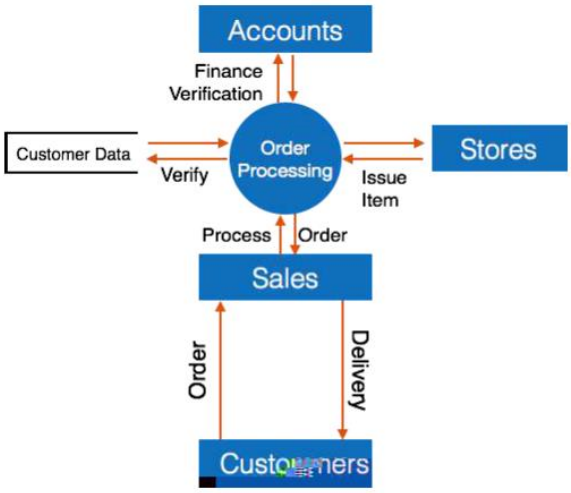
Process: any process that changes the data, producing an output. It might perform computations, or sort data based on logic, or direct the data flow based on business rules. A short label is used to describe the process, such as "Submit payment."

Data: store files or repositories that hold information for later use, such as a database table or a membership form. Each data store receives a simple label, such as "Orders."
Data flow: the route that data takes between the external entities, processes and data stores. It portrays the interface between the other components and is shown with arrows, typically labeled with a short data name, like "Billing details."
DFD Rules

- Each process should have at least one input and an output.
- Each data store should have at least one data flow in and one data flow out.
- Data stored in a system must go through a process.
- All processes in a DFD go to another process or a data store.

DFD Levels:

A data flow diagram can dive into progressively more detail by using levels and layers, zeroing in on a particular piece. DFD levels are numbered 0, 1 or 2, and occasionally go to even Level 3 or beyond.
Sample DFD:



Procedure:

1. Download and Install software tool like StarUML/Dia
2. Develop Level-1 and Level-2 DFD model.

Output:
Attach the snapshot of DFD Diagram.

Conclusion:
Insert your own conclusion for the experiment.

Experiment No. 06 – SRS Document

Aim:
To develop Software Requirement Specification (SRS) document in IEEE format for the selected case study.

Theory:

An SRS is basically an organization's understanding (in writing) of a customer or potential client's system requirements and dependencies at a particular point in time (usually) prior to any actual design or development work. It's a two-way insurance policy that assures that both the client and the organization understand the other's requirements from that perspective at a given point in time.

The SRS document itself states in precise and explicit language those functions and capabilities a software system must provide, as well as states any required constraints by which the system must abide. The SRS also functions as a blueprint for completing a project with as little cost growth as possible. The SRS is often referred to as the "parent" document because all subsequent project

management documents, such as design specifications, statements of work, software architecture specifications, testing and validation plans, and documentation plans, are related to it.
SRS should address the following:
a) Functionality. What is the software supposed to do?
b) External interfaces. How does the software interact with people, the system's hardware, other hardware, and other software?
c) Performance. What is the speed, response time, recovery time of various software functions, etc.?
d) Attributes. What are the portability, correctness, maintainability, security, etc. considerations?
e) Design constraints imposed on an implementation. Are there any required standards in effect, implementation language, policies for database integrity, operating environment(s) etc.?

Procedure:

- Prepare the SRS document as per the following guidelines:

A sample of basic SRS Outline

1. Introduction
1.1 Purpose
1.2 Document conventions
1.3 Intended audience
1.4 Contact information/SRS team members
1.5 References
2. Overall Description
2.1 Product perspective
2.2 Product features
2.3 User classes and characteristics
2.4 Operating environment
2.5 Design and Implementation constraints
2.6 User documentation
2.7 Assumptions and dependencies
3. External Interface Requirements
3.1 User interfaces
3.2 Hardware interfaces
3.3 Software interfaces
3.4 Communication interfaces
4. System Features
4.1 System feature A
4.1.1 Description and priority
4.1.2 Action/result
4.2 System feature B
5. Other Nonfunctional Requirements
5.1 Performance requirements
5.2 Safety requirements
5.3 Security requirements
5.4 Software quality attributes
6. Other Requirements
Appendix A: Terminology/Glossary/Definitions list
Appendix B: Analysis Model
Appendix C: Issues list

Output:
Attach your SRS document.

Conclusion:
Insert your own conclusion for the experiment.

Experiment No. 07 – LOC and FP Analysis

Aim:
To estimate the cost of project using software metrics (LOC and FP analysis).

Theory:

A software metric is a measure of software characteristics which are measurable or countable. Software metrics are valuable for many reasons, including measuring software performance, planning work items, measuring productivity, and many other uses. Within the software development process, many metrics are that are all connected. Software metrics are similar to the four functions of management: Planning, Organization, Control, or Improvement.

LOC Metrics:

It is one of the earliest and simpler metrics for calculating the size of the computer program. It is generally used in calculating and comparing the productivity of programmers. These metrics are derived by normalizing the quality and productivity measures by considering the size of the product as a metric.

Following are the points regarding LOC measures:

- In size-oriented metrics, LOC is considered to be the normalization value.
- It is an older method that was developed when FORTRAN and COBOL programming were very popular.
- Productivity is defined as KLOC / EFFORT, where effort is measured in person-months.
- Size-oriented metrics depend on the programming language used.
- As productivity depends on KLOC, so assembly language code will have more productivity.
- LOC measure requires a level of detail which may not be practically achievable.
- The more expressive is the programming language, the lower is the productivity.
- LOC method of measurement does not apply to projects that deal with visual (GUI-based) programming. As already explained, Graphical User Interfaces (GUIs) use forms basically. LOC metric is not applicable here.
- It requires that all organizations must use the same method for counting LOC. This is so because some organizations use only executable statements, some useful comments, and some do not. Thus, the standard needs to be established.
- These metrics are not universally accepted.

Advantages of LOC

- Simple to measure

Disadvantage of LOC

- It is defined on the code. For example, it cannot measure the size of the specification.
- It characterizes only one specific view of size, namely length, it takes no account of functionality or complexity
- Bad software design may cause an excessive line of code
- It is language dependent
- Users cannot easily understand it

Functional Point (FP) Analysis:

Allan J. Albrecht initially developed function Point Analysis in 1979 at IBM and it has been further modified by the International Function Point Users Group (IFPUG). FPA is used to make estimate of the software project, including its testing in terms of functionality or function size of the software product. The functional size of the product is measured in terms of the function point, which is a standard of measurement to measure the software application.

The basic and primary purpose of the functional point analysis is to measure and provide the software application functional size to the client, customer, and the stakeholder on their request. Further, it is used to measure the software project development along with its maintenance, consistently throughout the project irrespective of the tools and the technologies.

Following are the points regarding FPs

1. FPs of an application is found out by counting the number and types of functions used in the applications. Various functions used in an application can be put under five types, as shown:

Measurements Parameters	Examples
1. Number of External Inputs (EI)	Input screen and tables
2. Number of External Output (EO)	Output screens and reports
3. Number of external inquiries (EQ)	Prompts and interrupts.
4. Number of internal files (ILF)	Databases and directories
5. Number of external interfaces (EIF)	Shared databases and shared routines.

All these parameters are then individually assessed for complexity.

Sample:

Computing FPs

Measurement Parameter	Count		Weighing factor			
			Simple	Average	Complex	
1. Number of external inputs (EI)	—	*	3	4	6 =	—
2. Number of external Output (EO)	—	*	4	5	7 =	—
3. Number of external Inquiries (EQ)	—	*	3	4	6 =	—
4. Number of internal Files (ILF)	—	*	7	10	15 =	—
5. Number of external interfaces(EIF)	—	*	5	7	10 =	—
Count-total →						

"reliable backup and recovery required ?",
"data communication required ?",
"are there distributed processing functions ?",
"is performance critical ?",
"will the system run in an existing heavily utilized operational environment ?",
"on line data entry required ?",
"does the on line data entry require the input transaction to be built over multiple screens or operations ?",
"are the master files updated on line ?",
"is the inputs, outputs, files or inquiries complex ?",
"is the internal processing complex ?",
"is the code designed to be reusable ?",
"are the conversion and installation included in the design ?",
"is the system designed for multiple installations in different organizations ?",
"is the application designed to facilitate change and ease of use by the user ?"

The Function Point (FP) is thus calculated with the following formula:

FP = Count-total * [0.65 + 0.01 * F] where F is the sum of all 14 questionnaires on scale of 5
0 <= F <=70

Scale varies from 0 to 5 according to character of Complexity Adjustment Factor (CAF). Below table shows scale:

0 - No Influence 1 – Incidental 2 – Moderate 3 – Average
4 – Significant 5 - Essential

Procedure:

1. Find the cost of project using LOC.
2. Find the size of project using FP method.

Output:

Step wise calculation of project cost.

Conclusion:

Insert your own conclusion for the experiment.

Experiment No. 08 – Gantt Chart

Aim:

To track the project using Gantt chart

Theory:

Ganttproject/ProjectLibre is an open source framework used to perform planning, scheduling and resource allocation activities.

Task creation: First, you create some tasks by using the New Task button or directly from the Tasks menu choose New Task. The tasks appear on the tree on the left pane; you can directly change their name here. Next, you can organize tasks by indenting those forming groups or categories. Tasks can also be re-organized by using the up and down functions. These functions move the selected task up or down in its hierarchy reordering it.
Relationships: Ganttproject allows you to specify a relationship between two tasks. You can set them by dragging directly on the chart. Click and hold on the first task and moving the cursor to the second task. An arrow will appear, following the mouse. Drag the arrowhead to the second task and release the mouse. The second task will be dependent on the first one.
Creating resources: A project is composed of tasks and people (or resources) who are assigned to each task. You can create resources on the Resources panel by specifying the name, the function and contact information (mail or phone by example).
Assign resources to tasks: A resource can be assigned to a task directly on the properties dialog box of the task. Select the third tabbed panel and choose the name of the resource you want to assign. Then, specify a unit for the resources

Procedure:

1. Download and Install Gantt Project/ProjectLibre software.
2. Develop Gantt chart for your project.

Output:
Attach the snapshot of Gantt chart in pdf file.

Conclusion:
Insert your own conclusion for the experiment.

Experiment No. 09 – White Box Testing

Aim:
To write test cases for white box testing.

Theory:
White Box Testing is software testing technique in which internal structure, design and coding of software are tested to verify flow of input-output and to improve design, usability and security. In white box testing, code is visible to testers so it is also called Clear box testing, open box testing, transparent box testing, Code-based testing and Glass box testing. White box testing in software engineering is based on the inner workings of an application and revolves around internal testing.
Advantages of White Box Testing:

- Code optimization by finding hidden errors.
- White box tests cases can be easily automated.
- Testing is more thorough as all code paths are usually covered.
- Testing can start early in SDLC even if GUI is not available.

Disadvantages of White Box Testing:

- White box testing can be quite complex and expensive.
- Developers who usually execute white box test cases detest it. The white box testing by developers is not detailed can lead to production errors.
- White box testing requires professional resources, with a detailed understanding of programming and implementation.
- White-box testing is time-consuming, bigger programming applications take the time to test fully.

In this technique, all the possible combinations of the possible outcomes of conditions are tested at least once. Let's consider the following example:

```
READ X, Y
IF(X == 0 || Y == 0)
PRINT '0'
#TC1: X = 0, Y = 0
#TC2: X = 0, Y = 5
#TC3: X = 55, Y = 0
```

#TC4: X = 55, Y = 5
Hence, four test cases required for two individual conditions.
Similarly, if there are n conditions then 2n test cases would be required.

Procedure:

1. Create at least 10 test cases for your selected project in the prescribed format:

Project Name:

Test Case Template

Test Case ID: Fun_10	**Test Designed by:** <Name>
Test Priority (Low/Medium/High): Med	**Test Designed date:** <Date>
Module Name: Google login screen	**Test Executed by:** <Name>
Test Title: Verify login with valid username and password	**Test Execution date:** <Date>
Description: Test the Google login page	

Pre-conditions: User has valid username and password
Dependencies:

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Navigate to login page	User= example@gmail.com	User should be able to login	User is navigated to	Pass	
2	Provide valid username	Password: 1234		dashboard with successful		
3	Provide valid password			login		
4	Click on Login button					

Post-conditions:
User is validated with database and successfully login to account. The account session details are logged in database.

- Test case ID: Unique ID is required for each test case. Follow some conventions to indicate the types of the test. For Example, 'TC_UI_1' indicating 'user interfaces test case #1'.
- Test priority (Low/Medium/High): This is very useful during test execution. Test priorities for business rules and functional test cases can be medium or higher, whereas minor user interface cases can be of a low priority. Testing priorities should always be set by the reviewer.
- Module Name: Mention the name of the main module or the sub-module.
- Test Designed By Name of the Tester.
- Test Designed Date: Date when it was written.
- Test Executed By: Name of the Tester who executed this test.
- Test Execution Date: Date when the test was executed.
- Test Title/Name: Test case title. For example, verify the login page with a valid username and password.
- Test Summary/Description: Describe the test objective in brief.
- Pre-conditions: Any prerequisite that must be fulfilled before the execution of this test case. List all the pre-conditions in order to execute this test case successfully.
- Dependencies: Mention any dependencies on other test cases or test requirements.
- Test Steps: List all the test execution steps in detail. Write test steps in the order in which they should be executed.
- Expected Result: What should be the system output after test execution? Describe the expected result in detail including the message/error that should be displayed on the screen.
- Post-condition: What should be the state of the system after executing this test case?
- Actual result: The actual test result should be filled after test execution. Describe the system behavior after test execution.
- Status (Pass/Fail): If the actual result is not as per the expected result, then mark this test as failed. Otherwise, update it as passed.
- Notes/Comments/Questions: If there are any special conditions to support the above fields, which can't be described above or if there are any questions related to expected or actual results then mention them here.

Conclusion:
Insert your own conclusion for the experiment.

Experiment No. 10 – Black Box Testing

Aim:
To write test cases for black box testing.

Theory:

Black Box Testing is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioral Testing. Test cases are created considering the specification of the requirements. These test cases are generally created from working descriptions of the software including requirements, design parameters, and other specifications.

For the testing, the test designer selects both positive test scenario by taking valid input values and adverse test scenario by taking invalid input values to determine the correct output. Test cases are mainly designed for functional testing but can also be used for non-functional testing. Test cases are designed by the testing team, there is not any involvement of the development team of software.

Testing techniques:

Decision Table Technique	Decision Table Technique is a systematic approach where various input combinations and their respective system behavior are captured in a tabular form. It is appropriate for the functions that have a logical relationship between two and more than two inputs.
Boundary Value Technique	Boundary Value Technique is used to test boundary values, boundary values are those that contain the upper and lower limit of a variable. It tests, while entering boundary value whether the software is producing correct output or not.
State Transition Technique	State Transition Technique is used to capture the behavior of the software application when different input values are given to the same function.
Cause-Effect Technique	Cause-Effect Technique underlines the relationship between a given result and all the factors affecting the result. It is based on a collection of requirements.
Equivalence Partitioning Technique	Equivalence partitioning is a technique of software testing in which input data divided into partitions of valid and invalid values, and it is mandatory that all partitions must exhibit the same behavior.

Procedure:

1. Create at least 5 test cases for your selected project in the prescribed format:

Test Code	Test Case	Test Steps	Expected Result	Actual Result	Pass/ Fail
Test 1	Check Facebook login functionality with logged in Facebook user	1. Go to registration page 2. Click on "Login with Facebook" button	The window is redirected to Facebook, then to homepage with showing the Facebook timeline name on the top right of the bar. The name and details also saved in database.	As expected	Pass

Conclusion:
Insert your own conclusion for the experiment.

Experiment No. 11 – Risk Management

Aim:
To prepare RMMM Plan

Theory:

Risk is an expectation of loss, a potential problem that may or may not occur in the future. It is generally caused due to lack of information, control or time. A possibility of suffering from loss in software development process is called a software risk. A risk management technique is usually seen in the software Project plan. This can be divided into Risk Mitigation, Monitoring, and Management Plan (RMMM). In this plan, all works are done as part of risk analysis. As part of the overall project plan project manager generally uses this RMMM plan. In some software teams, risk is documented with the help of a Risk Information Sheet (RIS). After documentation of RMMM and start of a project, risk mitigation and monitoring steps will start.

Risk Mitigation:

It is an activity used to avoid problems (Risk Avoidance).

- Finding out the risk
- Removing causes that are the reason for risk creation.
- Controlling the corresponding documents from time to time.
- Conducting timely reviews to speed up the work.

Risk Monitoring:

It is an activity used for project tracking.

It has the following primary objectives as follows.

- To check if predicted risks occur or not.
- To collect data for future risk analysis.
- To allocate what problems are caused by which risks throughout the project.

Risk Management and planning:

It assumes that the mitigation activity failed and the risk is a reality. This task is done by Project manager when risk becomes reality and causes severe problems. If the project manager effectively uses project mitigation to remove risks successfully then it is easier to manage the risks. This shows that the response that will be taken for each risk by a manager.

Drawbacks of RMMM:

- It incurs additional project costs.
- It takes additional time.
- For larger projects, implementing an RMMM may itself turn out to be another tedious project.
- RMMM does not guarantee a risk-free project; in fact, risks may also come up after the project is delivered.

Risk	Category	Probability	Impact	RMMM plan
R1	Product Size	More	High	S1
R2	Customer	More	High	S2
R3	Customer	Less	Low	S3
R4	Execution	Less	High	S4
R5	Technical	Less	Low	S5
R6	Development	Less	Low	S6,S7
R7	Development	Less	High	S8

Risk information sheet			
Risk ID: P02-4-32	Date: 5/9/02	Prob: 80%	Impact: high
Description: Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.			
Refinement/context: Subcondition 1: Certain reusable components were developed by a third party with no knowledge of internal design standards. Subcondition 2: The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components. Subcondition 3: Certain reusable components have been implemented in a language that is not supported on the target environment.			
Mitigation/monitoring: 1. Contact third party to determine conformance with design standards. 2. Press for interface standards completion; consider component structure when deciding on interface protocol. 3. Check to determine number of components in subcondition 3 category; check to determine if language support can be acquired.			
Management/contingency plan/trigger: *RE* computed to be $20,200. Allocate this amount within project contingency cost. Develop revised schedule assuming that 18 additional components will have to be custom built; allocate staff accordingly. Trigger: Mitigation steps unproductive as of 7/1/02			
Current status: 5/12/02: Mitigation steps initiated.			
Originator: D. Gagne		Assigned: B. Laster	

Procedure:
1. Create RMMM/RIS of at least 5 risks that are involved in your mini project in the prescribed format.

Conclusion:
Insert your own conclusion for the experiment.

Assignment No. 01 – Software Design

Aim:
To develop an architectural design and mention the types of coupling and cohesion.

Theory:
IEEE defines architectural design as "the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system." The software that is built for computer-based systems can exhibit one of these many architectural styles. Each style will describe a system category that consists of:

- A set of components (e.g.: a database, computational modules) that will perform a function required by the system.
- The set of connectors will help in coordination, communication, and cooperation between the components.
- Conditions that how components can be integrated to form the system.
- Semantic models that help the designer to understand the overall properties of the system.

Coupling: Coupling is the measure of the degree of interdependence between the modules. A good software will have low coupling.
Types of Coupling:
Data Coupling: If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled. In data coupling, the components are independent of each other and communicate through data. Module communications don't contain tramp data. Example - customer billing system.

Stamp coupling: In stamp coupling, the complete data structure is passed from one module to another module. Therefore, it involves tramp data. It may be necessary due to efficiency factors- this choice was made by the insightful designer, not a lazy programmer.
Control Coupling: If the modules communicate by passing control information, then they are said to be control coupled. It can be bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality. Example - sort function that takes comparison function as an argument.
External Coupling: In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware. Ex- protocol, external file, device format, etc.
Common Coupling: The modules have shared data such as global data structures. The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change. So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses, and reduced maintainability.
Content Coupling: In a content coupling, one module can modify the data of another module, or control flow is passed from one module to the other module. This is the worst form of coupling and should be avoided.
Cohesion: Cohesion is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component.
Types of Cohesion:

Functional Cohesion: Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. It is an ideal situation.
Sequential Cohesion: An element outputs some data that becomes the input for other element, i.e., data flow between the parts. It occurs naturally in functional programming languages.
Communicational Cohesion: Two elements operate on the same input data or contribute towards the same output data. Example- update record in the database and send it to the printer.
Procedural Cohesion: Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable. Ex- calculate student GPA, print student record, calculate cumulative GPA, print cumulative GPA.
Temporal Cohesion: The elements are related by their timing involved. A module connected with temporal cohesion all the tasks must be executed in the same time span. This cohesion contains the code for initializing all the parts of the system. Lots of different activities occur, all at unit time.
Logical Cohesion: The elements are logically related and not functionally. Ex- A component reads inputs from tape, disk, and network. All the code for these functions is in the same component. Operations are related, but the functions are significantly different.
Coincidental Cohesion: The elements are not related (unrelated). The elements have no conceptual relationship other than location in source code. It is accidental and the worst form of cohesion. Ex - print next line and reverse the characters of a string in a single component.

Procedure:
Create a PPT showing an architectural design of your project and provide examples for various types of coupling and cohesion.

Output:
Attach the PPT.

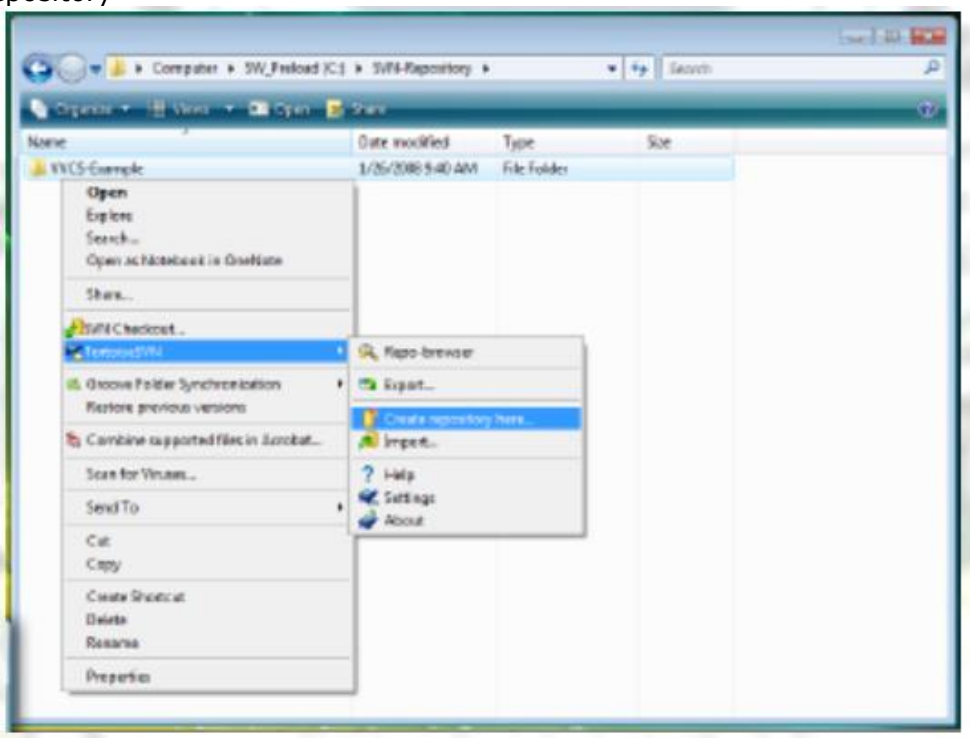
Assignment No. 02 – Software Configuration Management

Aim:
To perform Version control using Subversion.

Theory:
Version control tracks changes to source code or any other files. A good version control system can tell you what was changed, who changed it, and when it was changed. It allows a software developer to undo any changes to the code, going back to any prior version, release, or date. This can be particularly helpful when a researcher is trying to reproduce results from an earlier paper or report and merely requires documentation of the version number. Version control also provides a mechanism

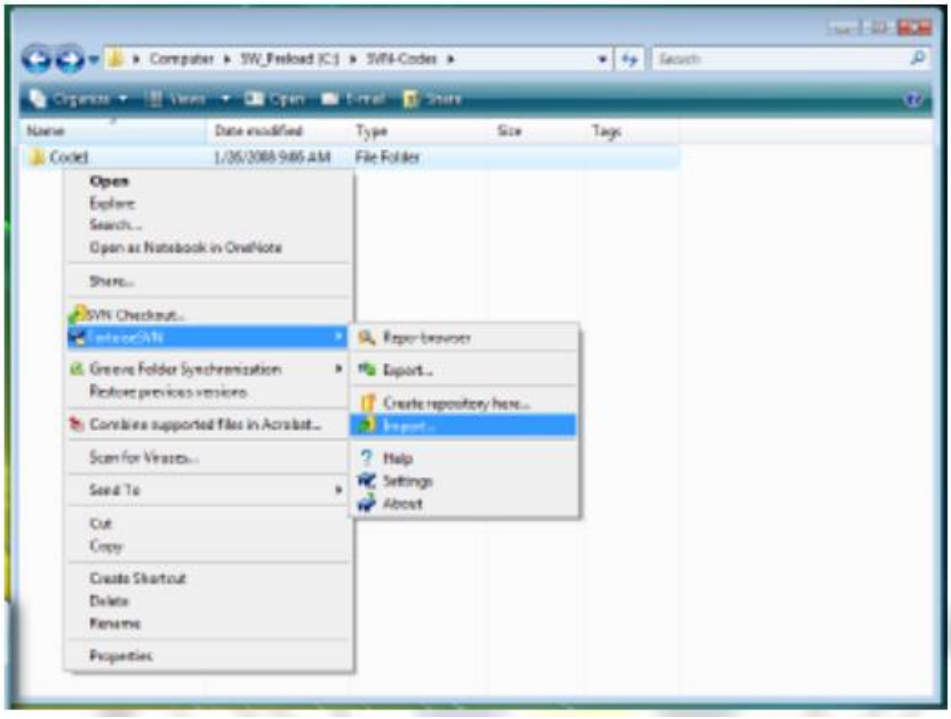
for incorporating changes from multiple developers, an essential feature for large software projects or any projects with geographically remote developers.
repository – single location where the current and all prior versions of the files are stored
working copy – the local copy of a file from the repository which can be modified and then checked in or "committed" to the repository
check-out – the process of creating a working copy from the repository (either the current version or an earlier version)
check-in – a check-in or commit occurs when changes made to a working copy are merged into the repository
diff – a summary of the differences between a working copy and a file in the repository, often taking the form of the two files side-by-side with differences highlighted
conflict – a conflict occurs when two or more developers attempt to make changes to the same file and the system is unable to reconcile the changes
update – merges recent changes to the repository into a working copy

The basic steps that one would use to get started with a version control tool are as follows:
1. Creating a Repository

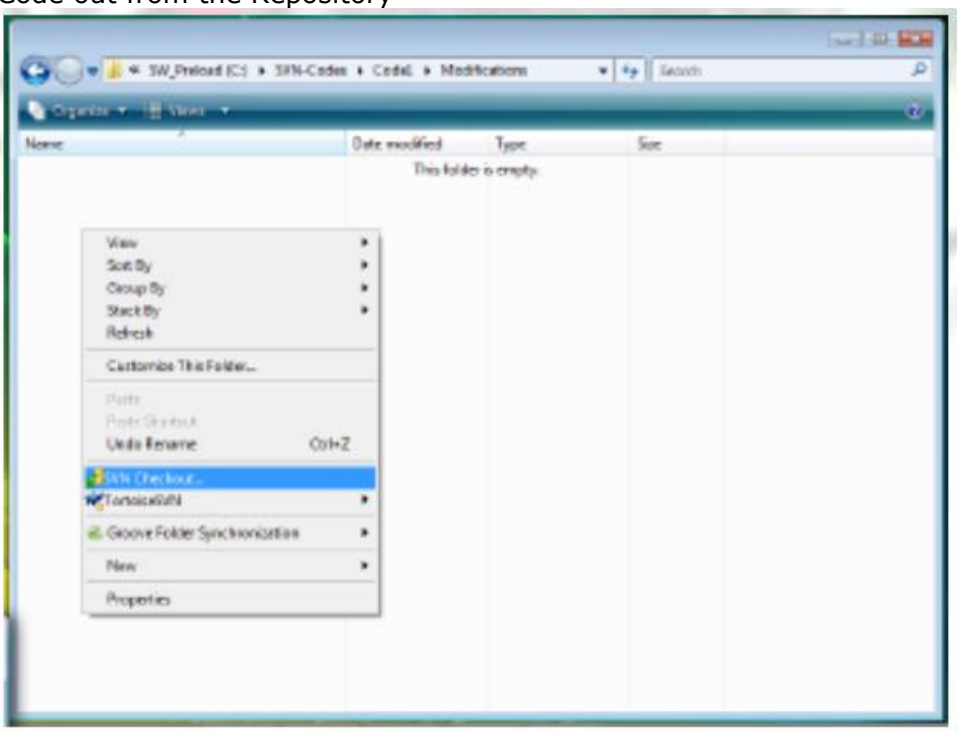


Determine a location for the repository, ideally on a server which is automatically backed up. Create a folder with the name of the repository; in this example the repository is called "VVCS- Example." Right click on the folder name, choose "TortoiseSVN" (which is integrated into the Microsoft Windows Explorer menu), then "Create Repository Here." Choose the Native File system, then you should see the message "Repository Successfully Created."

2. Importing a File into the Repository
Right click on the directory containing the file(s) and/or directory structure you wish to import to the repository (note, the directory that you click on will not be imported). Here we will simply be importing the file "code1.f" from directory "Code1." This code creates a 17×17 two-dimensional Cartesian grid for x and y between 0 and 1. Browse until you find the location of the repository "VVCS-Example" and select that directory name. This version of the code will be Revision 1.



3. Checking the Code out from the Repository

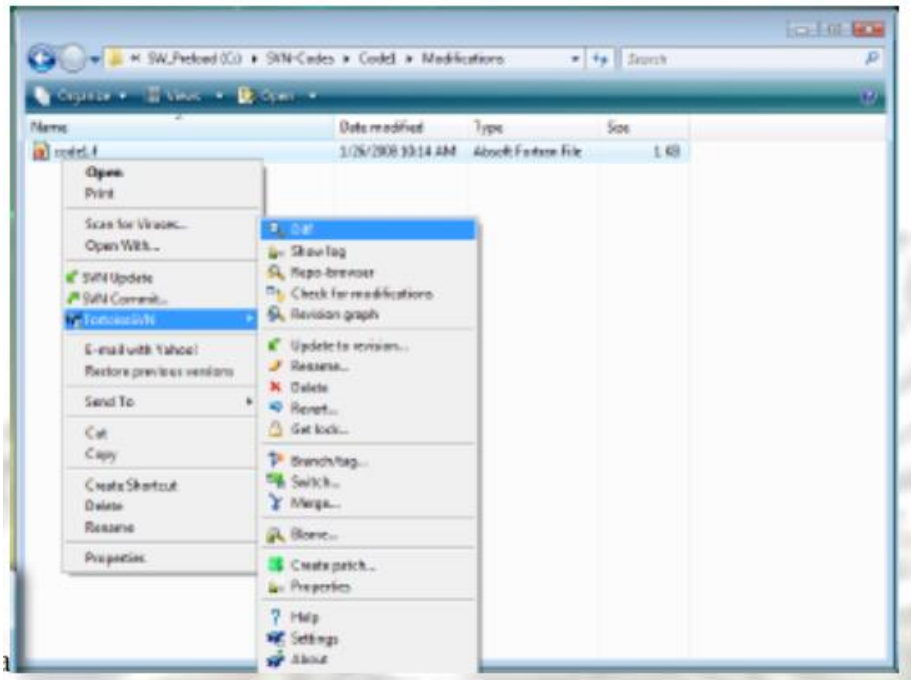


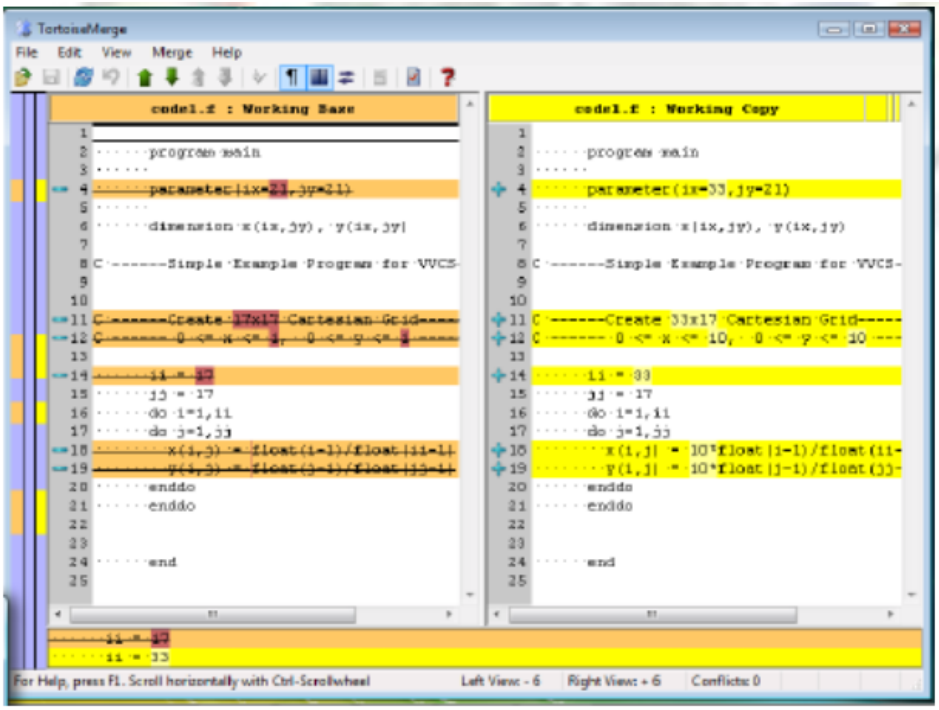
You now have the code "code1.f" safely placed in the repository. To modify this code and create a new revision, you will need to check out a working copy of the code. Go to the directory where you will be modifying the code, in this example, the directory "Modifications." Right click in Windows Explorer, and select "SVN Checkout…" Select the name of the repository you just created, then click "OK." You will now get a window telling you that you are at Revision 1. Notice the green check mark on the "code1.f" icon. This indicates that this working copy is up to date with the version in the repository.

4. Modify the Code and Compare to the Repository Version

The code "code1.f" can now be modified. Here we will change the code to allow the Cartesian grid to contain 33x17 points between the values of zero and ten. Once the code has been modified, you will notice that the green check mark has been replaced by a red exclamation point, indicating that the current working copy has been modified from the version in the repository. To examine these

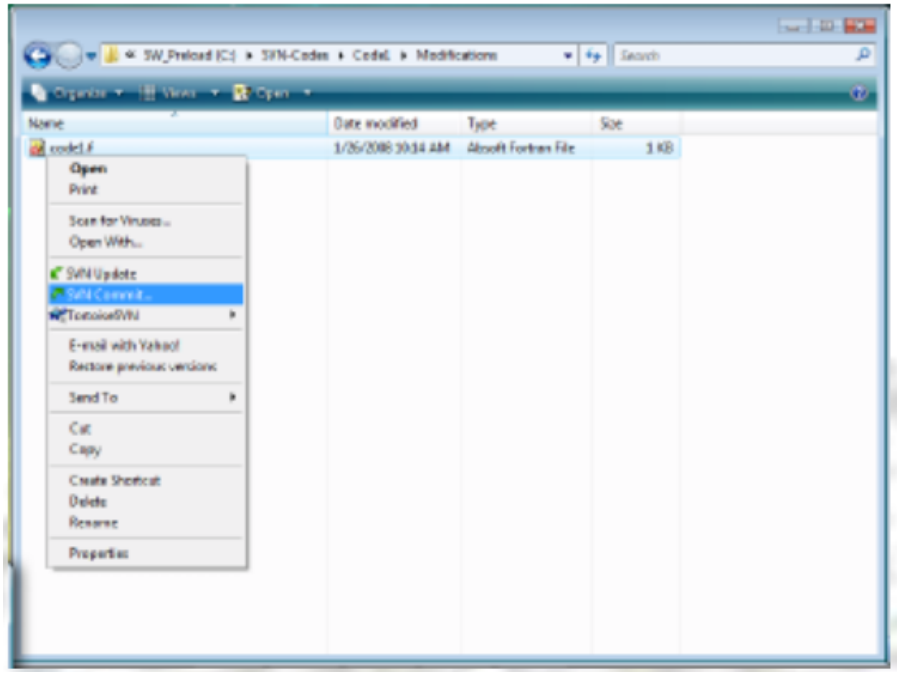
differences, right click on the "code1.f" file, select "TortoiseSVN," then "Diff." This opens the "TortoiseMerge" tool which clearly shows the modifications to the repository version (Working Base) that were made in the Working Copy.





5. Check the Code back into the Repository

When the changes are complete, the repository can be updated with your modified Working Copy by performing a check-in. Just right click on the file (or directory) and select "SVN Commit…" Enter a message describing the changes that were made, then select "OK." You are now at Revision 2.



Procedure:
Install TortoiseSVN software and synchronize your project. Attach the output snapshot.

Output:
Attach the snapshot of version control.