UNIT – I

Topic: - Introduction to Software Engineering

Unit-01/Lecture-01

Software Engineering : [**RGPV/JUNE-2013(7), JUNE 2011(10)]**

Practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate, and maintain them.

The study of systematic and effective processes and technologies for supporting software development and maintenance activities “”

– Improve quality”

– Reduce costs”

Software product may be

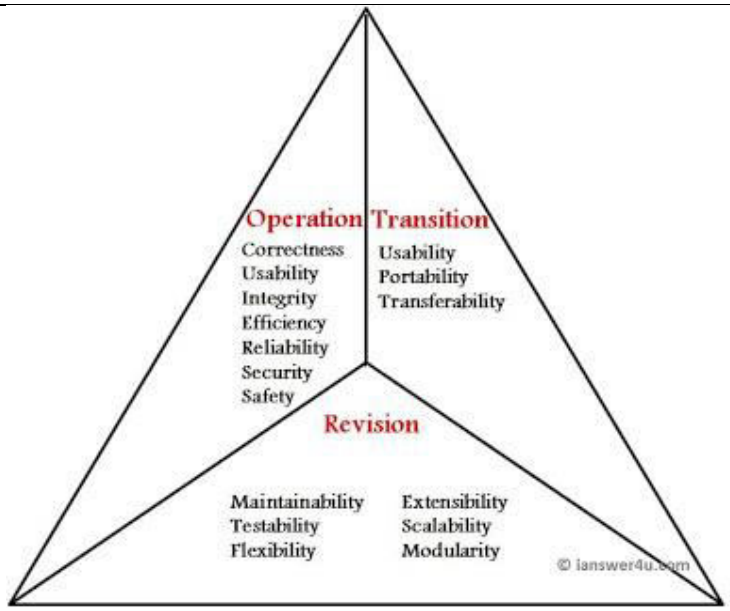
1. Custom- developed for a single customer according to their specifications.
2. Generic- developed to be sold to a range of different customers.

Software Characteristics

While developing any kind of software product, the first question in any developer’s mind is, “What are the qualities that good software should have?” Well before going into technical characteristics, I would like to state the obvious expectations one has from any software. First and foremost, a software product must meet all the requirements of the customer or end-user. Also, the cost of developing and maintaining the software should be low. The development of software should be completed in the specified time-frame.

Well these were the obvious things which are expected from any project (and software development is a project in itself). Now let’s take a look at Software Quality factors. These set of factors can be easily explained by Software Quality Triangle. The three characteristics of good application software are :-

1) Operational Characteristics
2) Transition Characteristics
3) Revision Characteristics



What Operational Characteristics should a software have ?

These are functionality based factors and related to 'exterior quality' of software. Various Operational Characteristics of software are :

a) Correctness: The software which we are making should meet all the specifications stated by the customer.

b) Usability/Learn ability: The amount of efforts or time required to learn how to use the software should be less. This makes the software user-friendly even for IT-illiterate people.

c) Integrity: Just like medicines have side-effects, in the same way a software may have a side-effect i.e. it may affect the working of another application. But a quality software should not have side effects.

d) Reliability: The software product should not have any defects. Not only this, it shouldn't fail while execution.

e) Efficiency: This characteristic relates to the way software uses the available resources. The software should make effective use of the storage space and execute command as per desired timing requirements.

f) Security: With the increase in security threats nowadays, this factor is gaining importance. The software shouldn't have ill effects on data / hardware. Proper measures should be taken to keep data secure from external threats.

g) Safety: The software should not be hazardous to the environment/life.

What are the Revision Characteristics of software?

These engineering based factors of relate to 'interior quality' of the software like efficiency, documentation and structure. These factors should be in-build in any good software. Various Revision Characteristics of software are :-

a) Maintainability: Maintenance of the software should be easy for any kind of user.
b) Flexibility: Changes in the software should be easy to make.
c) Extensibility: It should be easy to increase the functions performed by it.
d) Scalability: It should be very easy to upgrade it for more work(or for more number of users).
e) Testability: Testing the software should be easy.
f) Modularity: Any software is said to made of units and modules which are independent of each other. These modules are then integrated to make the final software. If the software is divided into separate independent parts that can be modified, tested separately, it has high modularity.

Transition Characteristics of the software :
a) Interoperability: Interoperability is the ability of software to exchange information with other applications and make use of information transparently.
b) Reusability: If we are able to use the software code with some modifications for different purpose then we call software to be reusable.
c) Portability: The ability of software to perform same functions across all environments and platforms, demonstrate its portability.

Importance of any of these factors varies from application to application. In systems where human life is at stake, integrity and reliability factors must be given prime importance. In any business related application usability and maintainability are key factors to be considered. Always remember in Software Engineering, quality of software is everything, therefore try to deliver a product which has all these characteristics and qualities.

Software Crisis:
A software crisis is a mismatch between what software can deliver and the capacities of computer systems, as well as expectations of their users. This became a growing problem in the 20^{th} century as computing grew by leaps and bounds and software was unable to keep pace. As the complexity of systems grows, so do the needs of users, who expect increasingly more performance from their software. Programmers may struggle to keep pace, creating a software crisis.

Consumer software typically moves through a slow series of development phases, but makes up a small portion of the volume of business in the industry. The bulk of software development is sunk into systems for specific applications, ranging from the programs that handle missile guidance aboard naval cruisers to internal record-keeping for health insurance companies. This software generally requires a substantial investment from the customer, as well as extensive programming from personnel charged with developing, testing, and maintaining it.

Such projects can run into a software crisis where they start to go over budget and take much longer than expected to develop. The programmers working on the software may have to deal with ongoing bug fixes while learning new aspects of a system, making adjustments for the client, and addressing other issues that arise. Low quality can be a concern, as the programmers may experience increasing pressure to meet budgets at all costs, even if it means the software won't be of good quality. Less documentation tends to be produced as well.

This is not just an issue for the development of new software products. Another concern can be the need to maintain older software which may have problems related to poor development or the failure to anticipate growing needs. Programmers could be spending large amounts of time on keeping legacy software functional so a company can continue to operate. With high investment in the older software, the company may be reluctant to order a new program, even if it would better meet their needs, because this could involve more expense and problems during the changeover.

Pressure to produce complex, advanced code can be a significant contributor to a software crisis. It can be difficult to control the pressure while keeping costs under control and staying on a time table. Some measures for dealing with a software crisis can include substantial advanced planning, selection of highly qualified personnel, and ongoing updates to make sure the project stays on task and on focus.

Traditional Software Engineering

One of the main events in the software industry at the beginning of the new century is the appearance of the agile methodologies for software development. Most of the companies today are using these methodologies to manage their projects. They radically changed the landscape of the development methods.

The agile methods are based on a set of new principals of design and build – different from the principals used before. However to understand the agile methodologies first of all we need to understand the reason of their appearance. Thus we've to first understand the traditional methods of software development – their features, their pros and cons, and nevertheless we've to understand their relevance.

Features of the Traditional Methods

The so called traditional approaches, also known as "engineering" approaches, are defined at the very beginning of the software sciences. Of course the software development process needs to be controlled somehow. To do so the software engineers came along with the well known engineering methods of controlling the processes. These methods are applying a disciplined approach where the stages of design and build are well predictable. Detailed stages of analysis and design precede the stage of building the software.

These methodologies are well documented and thus are quite complex to apply. Of course this is the main reason when we talk about their disadvantages. One of the main disadvantages is that these traditional methodologies are very bureaucratic. In practice the high level of detail in a methodology leads to a high level of complexity. Actually the work of managing the methodology itself is more than the work on the software product.

Traditional software engineering, CMMI and its problems

Well into the 1980s the largest buyer of software development services in the world, the US Department of Defense, was having trouble getting projects done on time, on budget and with the right specifications. Despite working with some of the best and most renowned software development companies out there, it still had close to a 50 / 50 chance of getting a project right. Worried about its performance, it set up to fund, together with Carnegie Mellon University, the Software Engineering Institute. This institute was tasked with compiling a set of software development best practices that could provide both a benchmark aginst which to measure current vendors, as well as concrete guidelines current vendors should follow to improve their software engineering practices. The initiative ultimately gave birth to the Capability Maturity Model Integration (CMMI) and its levels of 1 to 5.

The Capability Maturity Model ranks companies into levels, going from those that are at Level 1 (operating under processes that are not managed, or ad-hoc) to those that reach Level 5 (optimizing). It is not our purpose here to go into detail about what each level means, except to point out that at Level 5 companies must have complied with following the tenets of more than 16 software and systems development process areas, including the ability to (at level 2) manage their software development process (configuration management, project monitoring and control, project planning, requirements management); define their approach to software development (decision analysis and resolution, organizational training, risk management, validation, verification); quantitatively manage their development process (measure organizational process performance, reliably measure productivity, error injection and other key variables) and optimize their process, which involves constantly reviewing the causes of process problems and taking measures to improve and resolve them.

The nature of the DoD's needs influenced CMMI at the core. The DoD's software development projects tend to be large and often interact with hardware. Furthermore, they are government funded, which requires measures of "financial transparency" that limit budget flexibility, often requiring projects to provide a fixed cost. Thus, it is not surprising to see a CMMI model that was slanted towards traditional RUP (the Rational Unified Process) methodologies, especially in what regards to:

1. **Big Requirements Up Front** (BRUF). BRUF is necessary to adopt so-called formal estimation techniques (such as COCOMO I/II, Function Points, Feature Points, etc.) that allow a software development team to have a shot at estimating, in advance, the effort required –and hence the cost—of a project;

2. **High formality and preciseness in the way requirements are elicited and documented**. Under strict RUP or even under a more archaic "waterfall approach", requirement engineering is a ceremonious process that documents, in writing, use cases (i.e. using UML diagrams, for example), functional and non-functional requirements, process flows, mock-up screens, etc.
   It is not surprising that under RUP (or also under a traditional waterfall approach), the inception and elaboration stages take 40 to 50% of the total time spent in the software development project.

3. Testing "after the fact", i.e. after code has been written. This is clearly a quality control, but often leads to detecting errors late, when correcting them is more expensive.

4. Strict controls to "requirement changes". Processes for changing requirements are often so heavy and difficult, they seem to be designed more to prevent change than to manage change within a software development.

In essence, the **software development** methodologies proclaimed to be "best-practices" under CMMI and RUP, did produce significant gains in the quality of the code. However, this came at a significant price both in terms of:

- **Time** (time was more predictable but the formality **significantly increased the length of the software development** vs. ad-hoc programming);

- **Functional relevance** (a rigid process **did not allow applications to easily adapt to changes** in the business environment, so the ensuing application often was "what the client asked for but not really what the client needed";

- **Frustration with cost** (the ensuing application **was not cheap to make**, due to the administrative overhead of all the formality of the process, and the predicted cost of the application still did not match the initial "fixed price" estimate –yes, estimates where off by a lesser amount vs. ad-hoc estimations, but they were still significantly off.

S.NO	RGPV QUESTIONS	Year	Marks
Q.1	Discuss the major differences between software engineering & some other engineering. Discipline, such as bridge design or house building. Would you consider state of art software engineering as a true engineering discipline?	JUNE 2013	7
Q.2	What is prime objective of software engineering? Define software engineering paradigm.	JUNE 2011	10
Q.3	Give the importance of software engineering. Define software process. State the important features of a process	JUNE 2011	10

UNIT – I

Topic:-Object Oriented Software Engineering, Layered Technology

Unit-01/Lecture-02

Object Oriented Software Engineering

Object-oriented software engineering (commonly known by acronym OOSE) is an object modelling language and methodology.

OOSE was developed by Ivar Jacobson in 1992 while at Objectory AB. It is the first object-oriented design methodology to employ use cases to drive software design. It also uses other design products similar to those used by Object-modelling technique.

It was documented in the 1992 book Object-Oriented Software Engineering: A Use Case Driven Approach, ISBN 0-201-54435-0

The tool Objectory was created by the team at Objectory AB to implement the OOSE methodology. After success in the marketplace, other tool vendors also supported OOSE.

After Rational Software bought Objectory AB, the OOSE notation, methodology, and tools became superseded.

- As one of the primary sources of the Unified Modelling Language (UML), concepts and notation from OOSE have been incorporated into UML.
- The methodology part of OOSE has since evolved into the Rational Unified Process (RUP).
- The OOSE tools have been replaced by tools supporting UML and RUP.
- OOSE has been largely replaced by the UML notation and by the RUP methodology.

Software Engineering: Layered Technology-:-[RGPV/JUNE-2013(7)]

Software Development is a Layered Technology: Software development is totally a layered technology. That means, to develop software one will have to go from one layer to another. The layers are related and each layer demands the fulfillment of the previous layer. Figure below is the upward flowchart of the layers of software development.

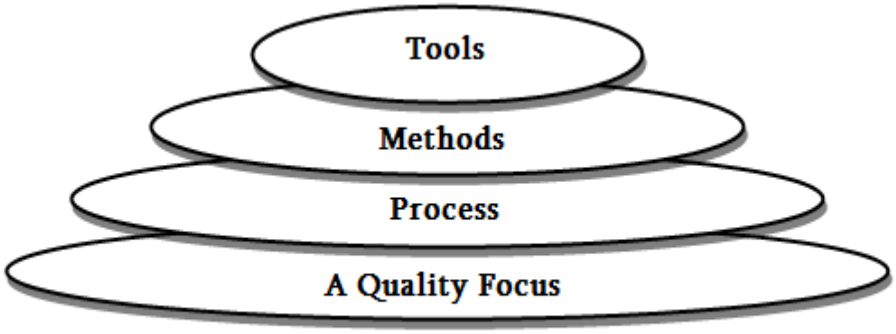


Figure: Flowchart of the Layers of Software Development

1. A Quality Focus : Software engineering must rest on an organizational commitment to quality. Total quality management and similar philosophies foster a continuous process improvement culture, and this culture ultimately leads to the development of increasingly more mature approaches to software engineering. The bedrock that supports software engineering is a quality focus.

2. Process: The foundation for software engineering is the process layer. Process defines a framework for a set of Key Process Areas (KPAs) that must be established for effective delivery of software engineering technology. This establishes the context in which technical methods are applied, work products such as models, documents, data, reports, forms, etc. are produced, milestones are established, quality is ensured, and change is properly managed.

3. Methods: Software engineering methods provide the technical how-to's for building software. Methods will include requirements analysis, design, program construction, testing, and support. This relies on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques.

3.Tools: Software engineering tools provide automated or semi-automated support for the process and the methods. When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer-aided software engineering, is established.

Goals of Software Engineering:

- Need to understand requirements.
- Want software with maximum functionality.
- Code must be reliable.
- Cost to develop and maintain important.
- Want results as fast as possible.
- Must minimize development risks**.**

Software Process Framework:

Software Process: Process defines a framework for a set of Key Process Areas (KPAs) that

must be established for effective delivery of software engineering technology. This establishes the context in which technical methods are applied, work products such as models, documents, data, reports, forms, etc. are produced, milestones are established, quality is ensured, and change is properly managed.

Software Process Framework: A process framework establishes the foundation for a complete software process by identifying a small number of framework activities that are applicable to all software projects, regardless of size or complexity. It also includes a set of umbrella activities that are applicable across the entire software process. Some most applicable framework activities are described below.

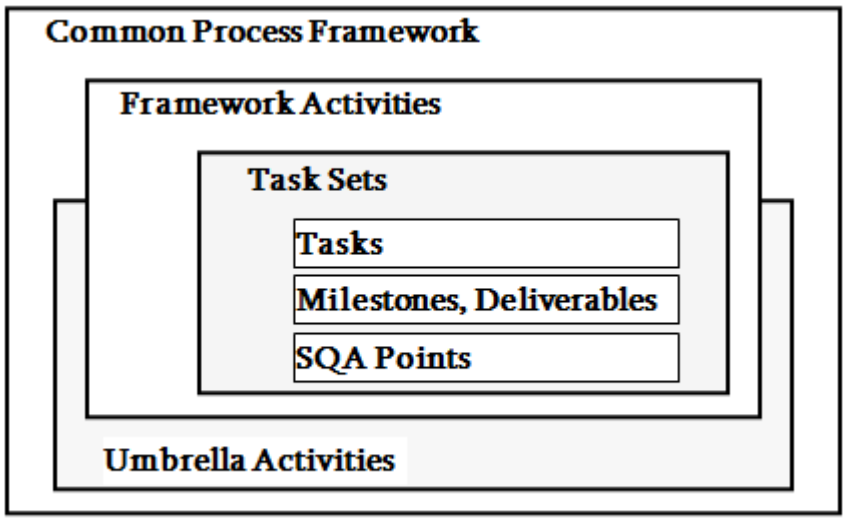


Figure: Chart of Process Framework

Communication: This activity involves heavy communication with customers and other stakeholders in order to gather requirements and other related activities.

Planning: Here a plan to be followed will be created which will describe the technical tasks to be conducted, risks, required resources, work schedule etc.

Modeling: A model will be created to better understand the requirements and design to achieve these requirements.

Construction: Here the code will be generated and tested.

Deployment: Here, a complete or partially complete version of the software is represented to the customers to evaluate and they give feedbacks based on the evaluation.

These above described five activities can be used in any kind of software development. The details of the software development process may become a little different, but the framework activities remain the same.

S.NO	RGPV QUESTIONS	Year	Marks
Q.1	What do you understand by a layered software design? What are the advantages of a layered design? Explain your answer by using suitable examples.	JUNE 2013	7

UNIT – I

Topic:-CBSE, Software Development Process

Unit-01/Lecture-03

Component Based Software Engineering[**RGPV/ JUNE-2014,2013(7)]**

Component-based software engineering (CBSE) (also known as component-based development (CBD)) is a branch of software that emphasizes the separation of concerns in respect of the wide-ranging functionality available throughout a given software system. It is a reuse-based approach to defining, implementing and composing loosely coupled independent components into systems. This practice aims to bring about an equally wide-ranging degree of benefits in both the short-term and the long-term for the software itself and for organizations that sponsor such software.

Software engineering practitioners regard components as part of the starting platform for service-orientation. Components play this role, for example, in web services, and more recently, in service-oriented architectures (SOA), whereby a component is converted by the web service into a *service* and subsequently inherits further characteristics beyond that of an ordinary component.

Components can produce or consume events and can be used for event-driven architectures (EDA).

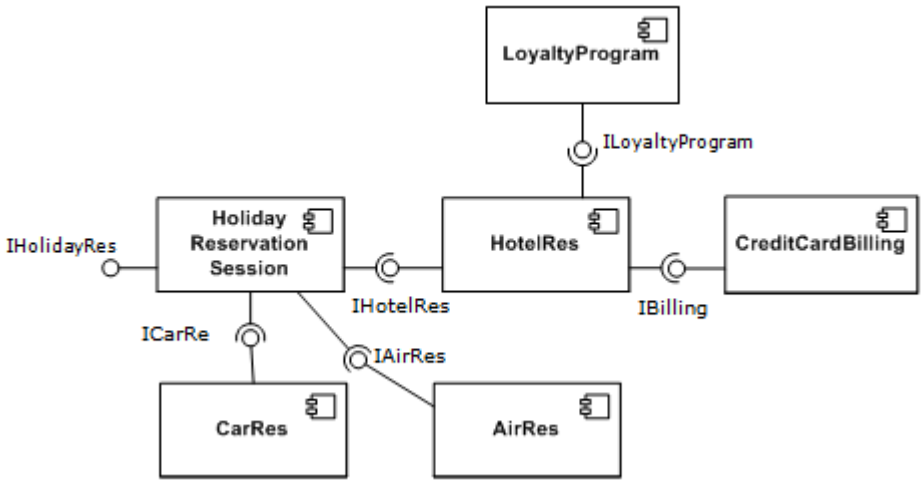
Characteristics of components

An individual software component is a software package, a web service, a web resource, or a module that encapsulates a set of related functions (or data).

All system processes are placed into separate components so that all of the data and functions inside each component are semantically related (just as with the contents of classes). Because of this principle, it is often said that components are modular and cohesive.

With regard to system-wide co-ordination, components communicate with each other via interfaces. When a component offers services to the rest of the system, it adopts a provided interface that specifies the services that other components can utilize, and how they can do so. This interface can be seen as a signature of the component – the client does not need to know about the inner workings of the component (implementation) in order to make use of it. This principle results in components referred to as encapsulated. The UML illustrations within this article represent provided interfaces by a lollipop-symbol attached to the outer edge of the component.

However, when a component needs to use another component in order to function, it adopts a used interface that specifies the services that it needs. In the UML illustrations in this article, used interfaces are represented by an open socket symbol attached to the outer edge of the component.



A simple example of several software components – pictured within a hypothetical holiday-reservation system represented in UML 2.0.

Another important attribute of components is that they are substitutable, so that a component can replace another (at design time or run-time), if the successor component meets the requirements of the initial component (expressed via the interfaces). Consequently, components can be replaced with either an updated version or an alternative without breaking the system in which the component operates.

As a general rule of thumb for engineers substituting components, component B can immediately replace component A, if component B provides at least what component A provided and uses no more than what component A used.

Software components often take the form of objects (not classes) or collections of objects (from object-oriented programming), in some binary or textual form, adhering to some interface (IDL) so that the component may exist autonomously from other components in a computer.

When a component is to be accessed or shared across execution contexts or network links, techniques such as serialization or marshalling are often employed to deliver the component to its destination.

Reusability is an important characteristic of a high-quality software component. Programmers should design and implement software components in such a way that many different programs can reuse them. Furthermore, component-based usability testing should be considered when software components directly interact with users.

It takes significant effort and awareness to write a software component that is effectively reusable. The component needs to be:

Software Development Life Cycle [**RGPV/ JUNE-2014,2012(7),JUNE 2011(10)]**

The Systems Development Life Cycle (SDLC), or Software Development Life Cycle in systems engineering, information systems and software engineering, is the process of creating or altering systems, and the models and methodologies that people use to

develop these systems. The concept generally refers to computer or information systems.

In software engineering the SDLC concept underpins many kinds of software development methodologies. These methodologies form the framework for planning and controlling the creation of an information system: the software development process.

The System Development Life Cycle framework provides a sequence of activities for system designers and developers to follow. It consists of a set of steps or phases in which each phase of the SDLC uses the results of the previous one.

A Systems Development Life Cycle (SDLC) adheres to important phases that are essential for developers, such as planning, analysis, design, and implementation, and are explained in the section below. A number of system development life cycle (SDLC) models have been created: waterfall, fountain, spiral, build and fix, rapid prototyping, incremental, and synchronize and stabilize. The oldest of these, and the best known, is the waterfall model: a sequence of stages in which the output of each stage becomes the input for the next. These stages can be characterized and divided up in different ways, including the following

There are following six phases in every Software development life cycle model:

Requirement gathering and analysis

Design

Implementation or coding

Testing

Deployment

Maintenance

1) Requirement gathering and analysis: Business requirements are gathered in this phase. This phase is the main focus of the project managers and stake holders. Meetings with managers, stake holders and users are held in order to determine the requirements like; Who is going to use the system? How will they use the system? What data should be input into the system? What data should be output by the system? These are general questions that get answered during a requirements gathering phase. After requirement gathering these requirements are analyzed for their validity and the possibility of incorporating the requirements in the system to be development is also studied.

Finally, a Requirement Specification document is created which serves the purpose of

guideline for the next phase of the model.

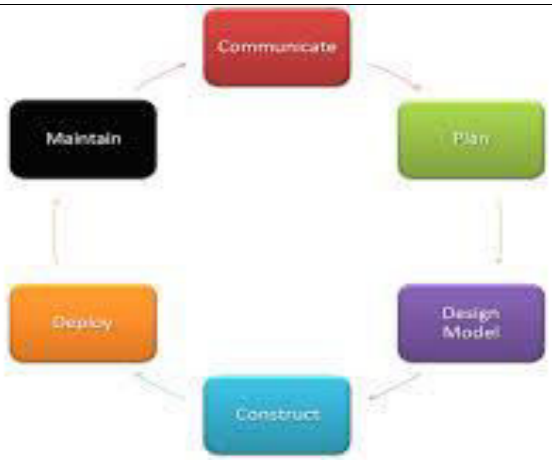
2) Design: In this phase the system and software design is prepared from the requirement specifications which were studied in the first phase. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture. The system design specifications serve as input for the next phase of the model.

3) Implementation / Coding: On receiving system design documents, the work is divided in modules/units and actual coding is started. Since, in this phase the code is produced so it is the main focus for the developer. This is the longest phase of the software development life cycle.

4) Testing**:** After the code is developed it is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase. During this phase unit testing, integration testing, system testing, acceptance testing are done.

5) Deployment: After successful testing the product is delivered / deployed to the customer for their use.

6) Maintenance: Once when the customers starts using the developed system then the actual problems comes up and needs to be solved from time to time. This process where the care is taken for the developed product is known as maintenance.

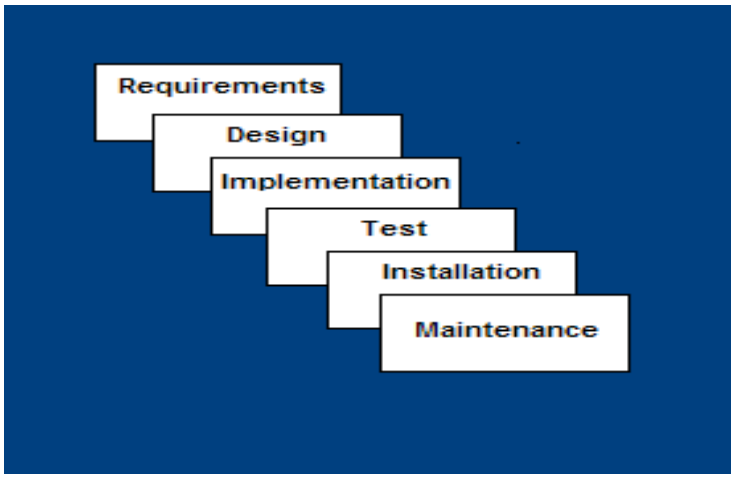


SDLC Model

1. Waterfall Model
2. V-Shaped Model
3. Evolutionary Prototyping Model
4. Spiral Method (SDM)
5. Iterative and Incremental Method
6. Extreme programming (Agile development)

Waterfall Model[RGPV/ JUNE-2011(5)]

This model involves finishing the first phase completely before commencing the next one. When each phase is completed successfully, it is reviewed to see if the project is on track and whether it is feasible to continue.



- **Requirements** – defines needed information, function, behavior, performance and interfaces.
- **Design** – data structures, software architecture, interface representations,

algorithmic details.

- **Implementation** – source code, database, user documentation, testing.
- Easy to understand, easy to use
- Provides structure to inexperienced staff
- Milestones are well understood
- Sets requirements stability
- Good for management control (plan, staff, track)
- Works well when quality is more important than cost or schedule

Waterfall Strength

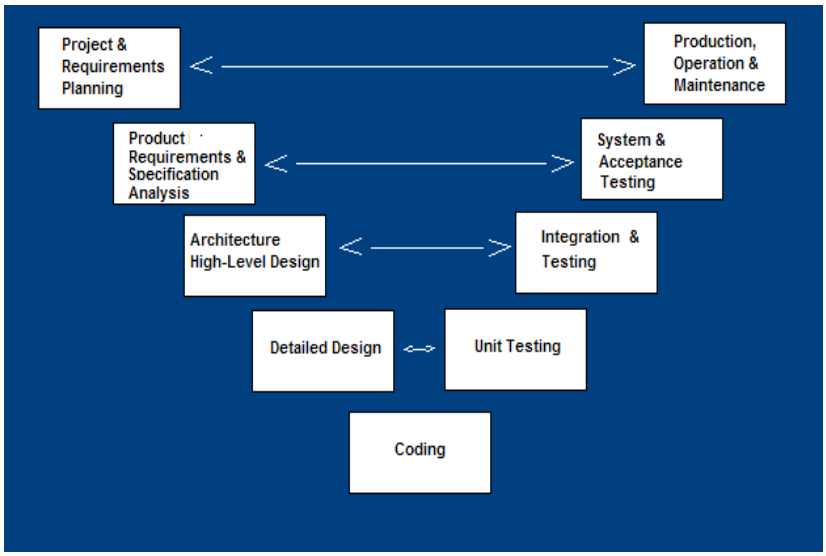
- All requirements must be known upfront
- Deliverables created for each phase are considered frozen – inhibits flexibility
- Can give a false impression of progress
- Does not reflect problem-solving nature of software development – iterations of phases
- Integration is one big bang at the end
- Little opportunity for customer to preview the system (until it may be too late)

Waterfall Deficiencies

- Requirements are very well known
- Product definition is stable
- Technology is understood
- New version of an existing product

V-Shaped Model:

This model focuses on execution of processes in a sequential manner, similar to the waterfall model but with more importance placed on testing. Testing procedures are written even before the commencement of writing code. A system plan is generated before starting the development phase.



V Shape Strength

- Emphasize planning for verification and validation of the product in early stages of product development
- Each deliverable must be testable
- Project management can track progress by milestones
- Easy to use

V Shape Weakness

- Emphasize planning for verification and validation of the product in early stages of product development
- Each deliverable must be testable
- Project management can track progress by milestones
- Easy to use

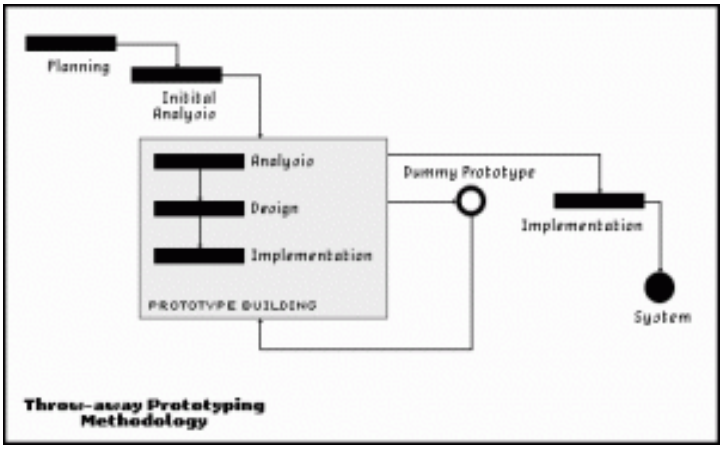
Evolutionary Prototyping Model

It refers to the activity of creating prototypes of software applications, for example, incomplete versions of the software program being developed. It is an activity that can occur in software development. It used to visualize some component of the software to limit the gap of misunderstanding the customer requirements by the development team. This also will reduce the iterations may occur in waterfall approach and hard to be implemented due to inflexibility of the waterfall approach. So, when the final prototype is developed, the requirement is considered to be frozen.

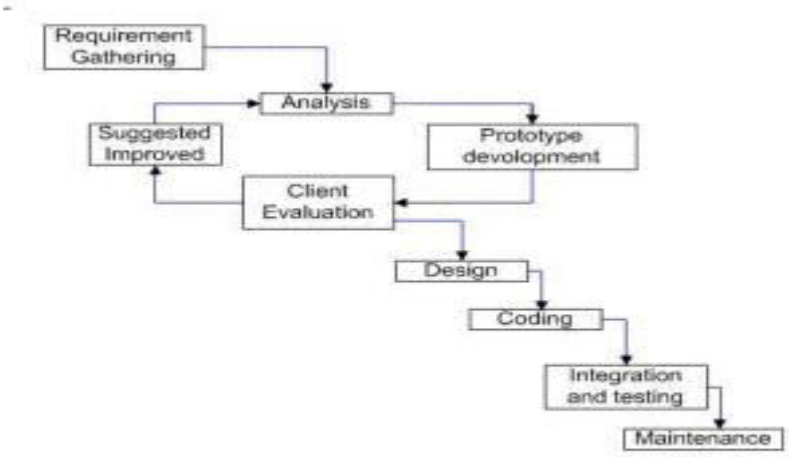
It has some types, such as:

· Throwaway prototyping: Prototypes that are eventually discarded rather than becoming a part of the finally delivered software

securing his information.

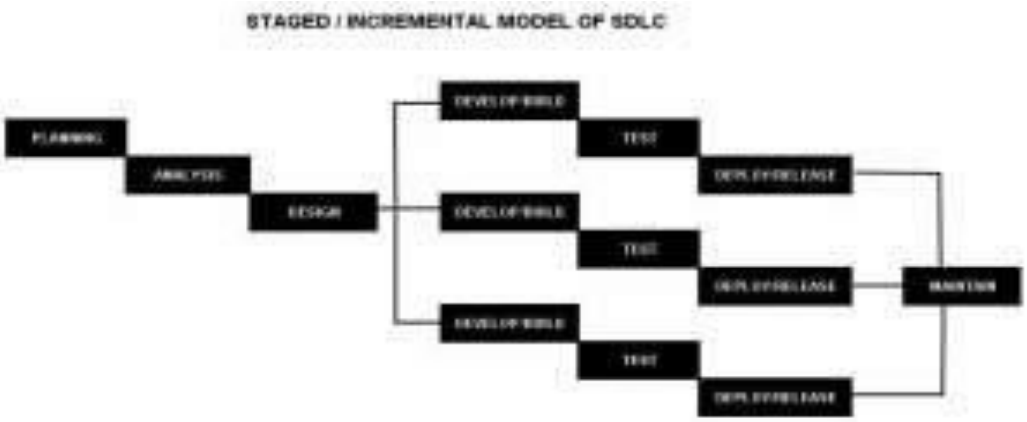


Evolutionary prototyping: prototypes that evolve into the final system through iterative incorporation of user feedback.



Evolutionary Prototyping Model

Incremental prototyping: The final product is built as separate prototypes. At the end the separate prototypes are merged in an overall design.



Extreme prototyping: used at web applications mainly. Basically, it breaks down web development into three phases, each one based on the preceding one. The first phase is a static prototype that consists mainly of HTML pages. In the second phase, the screens are programmed and fully functional using a simulated services layer. In the third phase the services are implemented

The usage

· This process can be used with any software developing life cycle model. While this shall be focused with systems needs more user interactions. So, the system do not have user interactions, such as, system does some calculations shall not have prototypes.

Advantages/Disadvantages

· Reduced time and costs, but this can be disadvantage if the developer lose time in developing the prototypes· Improved and increased user involvement.

· Insufficient analysis· User confusion of prototype and finished system.

· Developer misunderstanding of user objectives.

· Excessive development time of the prototype· Expense of implementing prototyping

S.NO	RGPV QUESTIONS	Year	Marks
Q.1	Explain the software life cycle model in detail.	JUNE 2013	7
Q.2	Explain Waterfall model.	JUNE 2011	5
Q.3	Define the term component. What are the benefits of component based software engineering (CBSE).	JUNE 2014, 2013	7

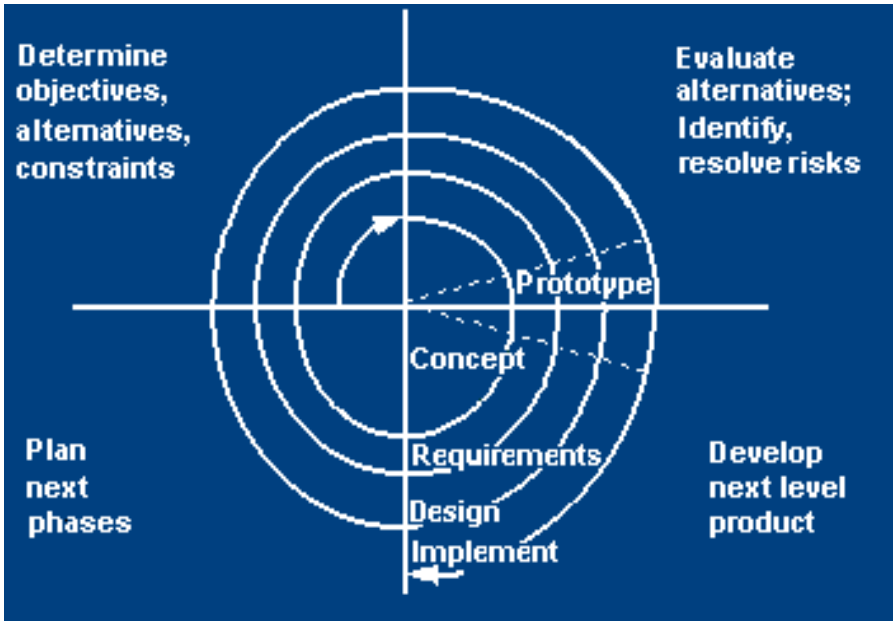
UNIT – I

Topic:-SDLC Model

Unit-01/Lecture-04

Spiral Model-:-[RGPV/JUNE-2013(10),JUNE-2011(5)]

- Adds risk analysis, and 4gl RAD prototyping to the waterfall model
- Each cycle involves the same sequence of steps as the waterfall process model



- Objectives: functionality, performance, hardware/software interface, critical success factors, etc.
- Alternatives: build, reuse, buy, sub-contract, etc.
- Constraints: cost, schedule, interface, etc.
- Spiral Quadrant
- Objectives: functionality, performance, hardware/software interface, critical success factors, etc.
- Alternatives: build, reuse, buy, sub-contract, etc.
- Constraints: cost, schedule, interface, etc.
- Objectives: functionality, performance, hardware/software interface, critical success factors, etc.
- Alternatives: build, reuse, buy, sub-contract, etc.
- Constraints: cost, schedule, interface, etc.

Spiral Model Strength

- Provides early indication of insurmountable risks, without much cost

- Users see the system early because of rapid prototyping tools
- Critical high-risk functions are developed first
- The design does not have to be perfect
- Users can be closely tied to all lifecycle steps
- Early and frequent feedback from users
- Cumulative costs assessed frequently

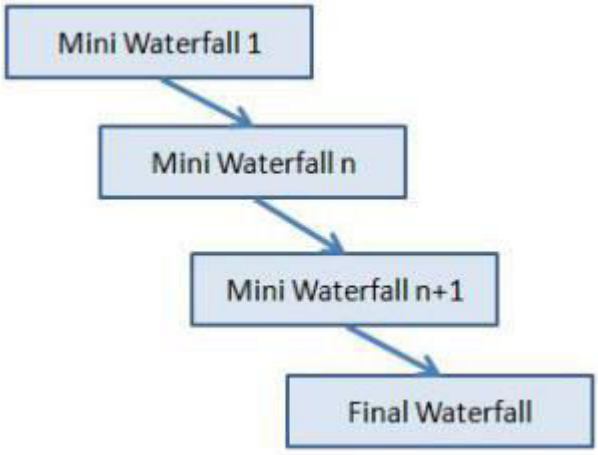
Spiral Model Weakness

- Time spent for evaluating risks too large for small or low-risk projects
- Time spent planning, resetting objectives, doing risk analysis and prototyping may be excessive
- The model is complex
- Risk assessment expertise is required
- Spiral may continue indefinitely
- Developers must be reassigned during non-development phase activities
- May be hard to define objective, verifiable milestones that indicate readiness to proceed through the next iteration

5. Iterative and Incremental Method

It is developed to overcome the weaknesses of the waterfall model. It starts with an initial planning and ends with deployment with the cyclic interactions in between. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental), allowing software developers to take advantage of what was learned during development of earlier parts or versions of the system.

It consists of mini waterfalls



The usage

It is used in shrink-wrap application and large system which built-in small phases or segments. Also can be used in system has separated components, for example, ERP system. Which we can start with budget module as first iteration and then we can start with inventory module and so forth.

Extreme programming (Agile development)

It is based on iterative and incremental development, where requirements and solutions evolve through collaboration between cross-functional teams.

The usage

It can be used with any type of the project, but it needs more involvement from customer and to be interactive. Also, it can be used when the customer needs to have some functional requirement ready in less than three weeks.

Advantages/Disadvantages

· Decrease the time required to avail some system features.· Face to face communication and continuous inputs from customer representative leaves no space for guesswork.· The end result is the high quality software in least possible time duration and satisfied customer · Scalability· Skill of the software developers· Ability of customer to express user needs· Documentation is done at later stages· Reduce the usability of components.· Needs special skills for the team.

S.NO	RGPV QUESTIONS	Year	Marks
Q.1	Explain the software life cycle model in detail.	JUNE 2013	7
Q.2	List several software process paradigms. Explain how both water fall model & prototype model can be accommodated in the spiral model?	JUNE 2014	7
Q.3	Explain why spiral model is considered to be meta data model.	JUNE 2013	10
Q.4	Discuss the selection process parameters for a life cycle model.	JUNE 2011	10

UNIT – I

Topic:-Rational Unified Model

Unit-01/Lecture-05

Rational Unified Model:- [**RGPVJUNE-2014,2013,2012(7)]**

THE UNIFIED APPROACH TO MODELING

The Unified Approach (UA) is the methodology for software development, based on methodologies by Booch, James Rumbaugh and Jacobson. UA has the following features:

1. Just like Jacobson's model, the UA is also a Use-case driven software development methodology. In UA to modeling, all the models are built around the use-case model.
2. UA utilizes a standard set of tools for modeling.
3. The OO analysis is done by utilizing use-cases and object modeling.
4. It favors repositories of reusable classes and emphasizes on maximum reuse.
5. UA follows a layered approach to software development.
6. UA is suitable for different development lifecycles such as Incremental development and prototyping.
7. UA favors continuous testing throughout the development lifecycle.

Unified process is a refinement of rational unified process. It is an extensible framework that can be customized for specific projects.

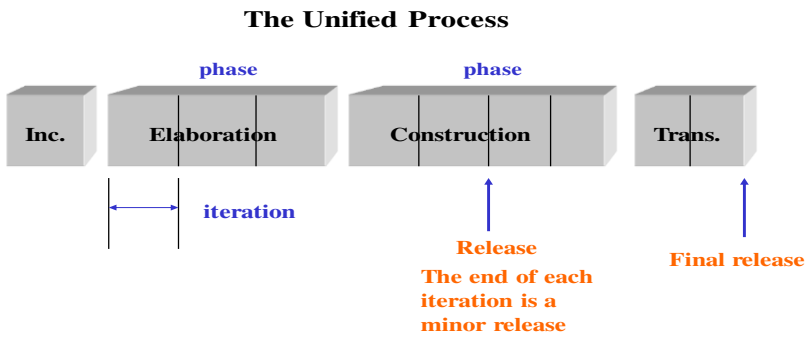
This process divides the development process into four phases:

- Inception
- Elaboration
- Conception
- Transition

UP have the following major characteristics:

- It is use-case driven
- It is architecture-centric
- It is risk focused
- It is iterative and incremental

The 6 RUP Best Practices



1.Develop Iteratively

The software requirements specification (SRS) keeps on evolving throughout the development process and loops are created to add them without affecting the cost of development.

2.Manage Requirements

The business requirements documentation and project management requirements need to be gathered properly from the user in order to reach the targeted goal.

3. Use Components

The components of large project which are already tested and are in use can be conveniently used in other projects. This reuse of components reduces the production time.

4. Model Visually

Use of Unified modeling language (UML) facilitates the analysis and design of various components. Diagrams and models are used to represent various components and their interactions.

5. Verify Quality

Testing and implementing effective project quality management should be a major part of each and every phase of the project from initiation to delivery (aka the project management life cycle).

6. Control Changes

Synchronization of various parts of the system becomes all the more challenging when the parts are being developed by various teams working from different geographic locations on different development platforms. Hence special care should be taken in this

direction so that the changes can be controlled.

Advantages of RUP Software Development

1. This is a complete methodology in itself with an emphasis on accurate documentation.
2. It is proactively able to resolve the project risks associated with the client's evolving requirements requiring careful change request management.
3. Less time is required for integration as the process of integration goes on throughout the software development life cycle.
4. The development time required is less due to reuse of components.
5. There is online training and tutorial available for this process.

Disadvantages of RUP Software Development

1.The team members need to be expert in their field to develop a software under this Methodology.
2. The development process is too complex and disorganized.
3. On cutting edge projects which utilize new technology, the reuse of components will not be possible. Hence the time saving one could have made will be impossible to fulfill.
4. Integration throughout the process of software development, in theory sounds a good thing. But on particularly big projects with multiple development streams it will only add to the confusion and cause more issues during the stages of testing

S.NO	RGPV QUESTIONS	Year	Marks
Q.1	What are the different phases of unified modeling?	JUNE 2012	10
Q.2	Explain the unified approach to software development. Discuss the merits & demerits of this approach.	JUNE 2013,2014	7

UNIT – I

Topic:- Difference between process & methodology

Unit-01/Lecture-06

Difference between process & methodology

Processes are probably the easiest to understand since we all work with them daily, even when we don't know it. A process is simply a well defined set of steps and decisions points for executing a specific task. Well planned and deeply understood processes are essential to your ability to automate business tasks. This is really where that magic happens, a process level. Generally speaking, processes are highly repeatable and if it can be repeated it can be automated (with exceptions or occasional human intervention of course).

An example of a process would be how you process payments to vendors you work with and the steps and decisions involved in doing so. When working through processes and defining them be EXTREMELY detailed, otherwise attempts to automate will end in futility. Capture every action and decision and outline sub-processes as necessary.
Hopefully that clears up the differences between frameworks, methodologies and processes. While they appear somewhat hierarchical, and can be, they aren't always necessarily so, particularly in both directions. Frameworks will typically have methodologies but not always. Methodologies will typically have [multiple] processes embedded, but some processes will be stand alone.

The real power of combining these things is in developing processes in the context of a methodology and applying methodologies in the context of a framework and most importantly, when you utilize all of those things in the context of YOUR business. The exact same approach won't work for every business but every business, no matter how large or small, can benefit from applying these principles.

software configuration management-:-[RGPV/JUNE-2014(7)]

Software configuration management is referred to as source control, change management, and version control.

Software configuration management systems are commonly used in software development groups in which several developers are concurrently working on a common set of files. If two developers change the same file, that file might be overwritten and critical code changes lost. Software configuration management systems are designed to avoid this inherent problem with sharing files in a multiuser environment.

Any software configuration management system creates a central repository to facilitate file sharing. Each file to be shared must be added to the central repository to create the

first version of the file. After a file is part of the central repository, users can access and update it, creating new versions.
Software Configuration Management (SCM) is the overall management of a software design project as it evolves into a software product or system. This includes technical aspects of the project, all level of communications, organization, and the control of modifications changes to the project plan by the programmers during the development phase. Software Configuration Management is also called Software Control Management.

Changes are inevitable when software is built. A primary goal of software engineering is to improve the ease with which changes can be made to software. Configuration management is all about change control. Every software engineer has to be concerned with how changes made to work products are tracked and propagated throughout a project. To ensure that quality is maintained the change process must be audited. A Software Configuration Management (SCM) Plan defines the strategy to be used for change management.

Software Configuration Items (SCI)

- Computer programs (both source and executable)
- Documentation (both technical and user)
- Data (contained within the program or external to it)

Fundamental Sources of Change

- New business or market conditions dictate changes to product requirements or business rules
- New stakeholder needs demand modification of data, functionality, or services delivered by a computer-based system
- Business reorganization causes changes in project priorities or software engineering team structure
- Budgetary or scheduling constraints cause system to be redefined

Configuration Management System Elements

- *Component elements* – set of tools coupled within a file management system to enable access to and management of each SCI
- *Process elements* – collection of procedures and tasks that define and effective approach to change management for all stakeholders
- *Construction elements* – set of tolls that automate the construction of software by ensure a set of validated components is assembled
- *Human elements* – team uses a set of tools and process features encompassing other CM elements

Baselines

- A work product becomes a baseline only after it is reviewed and approved.
- A baseline is a milestone in software development that is marked by the delivery of

one or more configuration items.

- Once a baseline is established each change request must be evaluated and verified by a formal procedure before it is processed.
- Baseline work products are place in a project database or repository

SCM Repository Functions

- Data integrity
- Information sharing
- Tool integration
- Data integration
- Methodology enforcement
- Document standardization

SCM Content

- Problem description
- Problem domain information
- Emerging system solution
- Software process rules and instructions
- Project plan, resources, and history
- Organizational content information

SCM Tool Features

- **Versioning** - control changes to all work products before and after release to customer
- **Dependency tracking and change management** - tracking relationships among multiple versions of work products to enable efficient changes (link management)
- **Requirements tracing** – depends on link management, provides the ability to track all work products that result from a specific requirements specification (forward tracing) and to identify which requirement generated by any given work product (backward tracing)
- **Configuration management** – works closely with link management and versioning facilities to keep track of a series of configurations representing project milestones or production releases
- **Audit trails** - establishes additional information about when, where, why, and by whom changes were made

SCM Process Objectives

1. Identify all items that define the software configuration
2. Manage changes to one or more configuration items
3. Facilitate construction of different versions of a software application
4. Ensure that software quality is maintained as configuration evolves

Software Configuration Management Tasks

- Identification (tracking multiple versions to enable efficient changes)
- Version control (control changes before and after release to customer)
- Change control (authority to approve and prioritize changes)
- Configuration auditing (ensure changes made properly)
- Reporting (tell others about changes made)

Configuration Object Identification

- To control and manage configuration items, each must be named and managed using an object-oriented approach
- Basic objects are created by software engineers during analysis, design, coding, or testing
- Aggregate objects are collections of basic objects and other aggregate objects
- Configuration object attributes: unique name, description, list of resources, and a realization (a pointer to a work product for a basic object or null for an aggregate object)

Version Control

- Combines procedures and tools to manage the different versions of configuration objects created during the software process
- Version control systems require the following capabilities
  - Project repository – stores all relevant configuration objects
  - Version management capability – stores all versions of a configuration object (enables any version to be built from past versions)
  - Make facility – enables collection of all relevant configuration objects and construct a specific software version
  - Issues (bug) tracking capability – enables team to record and track status of outstanding issues for each configuration object
- Uses a system modeling approach (template – includes component hierarchy and component build order, construction rules, verification rules)

Change Control

- Change request is submitted and evaluated to assess technical merit and impact on the other configuration objects and budget
- Change report contains the results of the evaluation
- Change control authority (CCA) makes the final decision on the status and priority of the change based on the change report
- Engineering change order (ECO) is generated for each change approved (describes change, lists the constraints, and criteria for review and audit)
- Object to be changed is checked-out of the project database subject to access control parameters for the object
- Modified object is subjected to appropriate SQA and testing procedures
- Modified object is checked-in to the project database and version control mechanisms

are used to create the next version of the software
- Synchronization control is used to ensure that parallel changes made by different people don't overwrite one another

Software Configuration Audit Questions

1. Has the change specified by the ECO been made without modifications?
2. Has a FTR been conducted to assess technical correctness?
3. Was the software process followed and software engineering standards been properly applied?
4. Do the attributes of the configuration object reflect the change?
5. Have the SCM standards for recording and reporting the change been followed?
6. Were all related SCI's properly updated?

Content Management System

- Establishes a process that acquires existing content, structures it to be presented to an end-user, and provides for display to the client-side environment
- Collection subsystem
  - converts content to displayable format
  - organizes content for client-side display
- Management subsystem
  - content database
  - database capabilities
  - configuration management functions
- Publishing subsystem
  - static elements
  - publication services
  - external services

Benefits of software configuration management

If you have not used a software configuration management system or are not that familiar with the concept, you might wonder whether it is appropriate to use software configuration management on your project. Test automation is a software development effort. Every time a test script is created, whether through recording or coding, a file is generated that contains code. When created, developed, or edited, that code is a valuable test asset.

A team environment presents the risk of losing functioning code or breaking test scripts by overwriting files. A software configuration management system offers a way to overcome this risk. Every time a file changes, a new version is created and the original file is preserved.

For the team that is new to software configuration management, all of the essential

features for versioning test scripts are available through the Functional Tester interface. This integration simplifies the use and adoption of software configuration management.

Software configuration management products

The ClearCase or Rational Team Concert integration for versioning Functional Tester test assets is specialized and cannot be duplicated with other tools. For this reason, some ClearCase operations cannot be performed outside Functional Tester.

When you use Functional Tester, the ClearCase or Rational Team Concert operations appear to be very simple. But a lot is going on behind the scenes. A Functional Tester script is a collection of files. The complexity of treating several files as a single entity is hidden because all actions in the Functional Tester user interface are performed on the script. You do not see the related files anywhere in the user interface. In addition, some software configuration management operations, such as merging, are very complex. There is built-in logic to determine the order in which files are merged, and then different utilities are employed as needed to complete the merge.

S.NO	RGPV QUESTIONS	Year	Marks
Q.1	Explain in detail configuration management.	JUNE 2014	7

UNIT – I

Topic:-Safety

Unit-01/Lecture-07

Safety in Software Engineering:-

Software system safety, an element of the total safety and software development program, cannot be allowed to function independently of the total effort. Both simple and highly integrated multiple systems are experiencing an extraordinary growth in the use of computers and software to monitor and/or control safety-critical subsystems or functions. A software error, design flaw, or the lack of generic safety-critical requirements can contribute to or cause a system failure or erroneous human decision. To achieve an acceptable level of safety for software used in critical applications, software system safety engineering must be given primary emphasis early in the requirements definition and system conceptual design process.

Safety-critical software must then receive continuous management emphasis and engineering analysis throughout the development and operational lifecycles of the system. Software system safety is directly related to the more critical design aspects and safety attributes in software and system functionality, whereas software quality attributes are inherently different and require standard scrutiny and development rigor. Software safety hazard analysis required for more complex systems where software is controlling critical functions generally are in the following sequential categories and are conducted in phases as part of the system safety or safety engineering process: software safety requirements analysis; software safety design analyses (top level, detailed design and code level); software safety test analysis, and software safety change analysis.

Once these "functional" software safety analyses are completed the software engineering team will know where to place safety emphasis and what functional threads, functional paths, domains and boundaries to focus on when designing in software safety attributes to ensure correct functionality and to detect malfunctions, failures, faults and to implement a host of mitigation strategies to control hazards. Software security and various software protection technologies are similar to software safety attributes in the design to mitigate various types of threats vulnerability and risks. Deterministic software is sought in the design by verifying correct and predictable behaviour at the system level.

Goals

- Safety consistent with mission requirements, is designed into the software in a timely, cost effective manner.

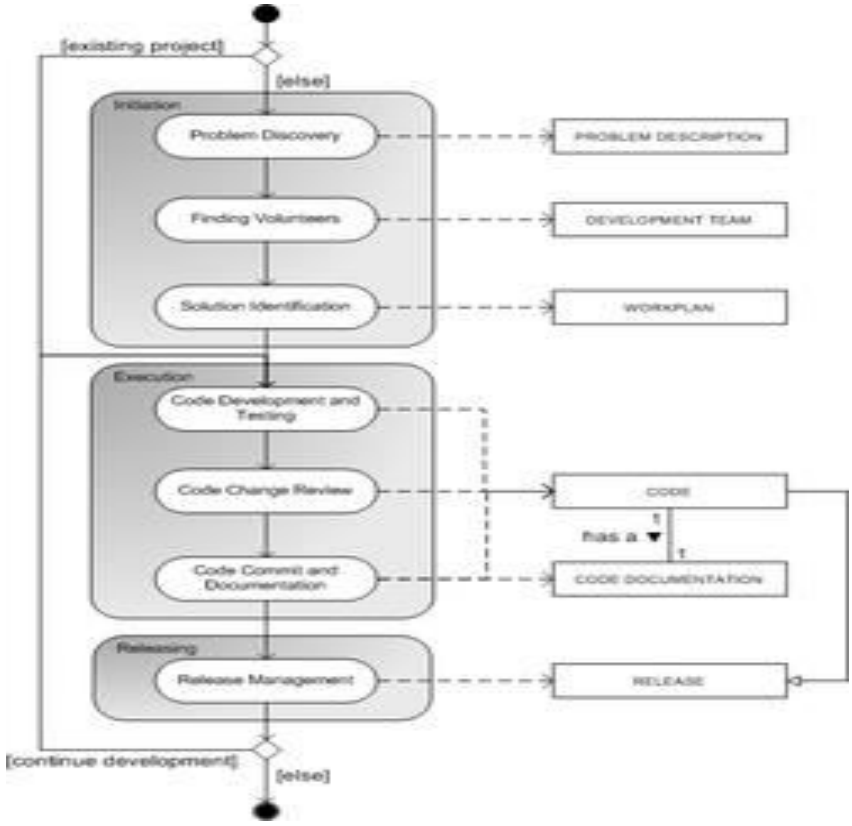
- On complex systems involving many interactions safety-critical functionality should be identified and thoroughly analyzed before deriving hazards and design safeguards for mitigations.
- Safety-critical functions lists and preliminary hazards lists should be determined proactively and influence the requirements that will be implemented in software.
- Contributing factors and root causes of faults and resultant hazards associated with the system and its software are identified, evaluated and eliminated or the risk reduced to an acceptable level, throughout the lifecycle.
- Reliance on administrative procedures for hazard control is minimized.
- The number and complexity of safety critical interfaces is minimized.
- The number and complexity of safety critical computer software components is minimized.
- Sound human engineering principles are applied to the design of the software-user interface to minimize the probability of human error.
- Failure modes, including hardware, software, human and system are addressed in the design of the software.
- Sound software engineering practices and documentation are used in the development of the software.
- Safety issues and safety attributes are addressed as part of the software testing effort at all levels.
- Software is designed for human machine interface, ease of maintenance and modification or enhancement
- Software with safety-critical functionality must be thoroughly verified with objective analysis and preferably test evidence that all safety requirements have been met per established criteria.

Open Source Software Development

Open-source software development is the process by which open-source software, or similar software whose source code is publicly available, is developed. These are software products available with its source code under an open-source license to study, change, and improve its design. Examples of some popular open-source software products are Mozilla Firefox, Google Chromium, Android, LibreOffice and the Apache OpenOffice Suite. Open-source software development has been a large part of the creation of the World Wide Web as we know it, with Tim Berners-Lee contributing his HTML code development as the original platform upon which the internet is now built.

Open-source software development model

Open-source software development can be divided into several phases. The phases specified here are derived from Sharma et al.[3] A diagram displaying the process-data structure of open-source software development is shown on the right. In this picture, the phases of open-source software development are displayed, along with the corresponding data elements. This diagram is made using the meta-modeling and meta-process modeling techniques.



Starting an open-source project

There are several ways in which work on an open-source project can start-

1. An individual who senses the need for a project announces the intent to develop a project in public.
2. A developer working on a limited but working codebase, releases it to the public as the first version of an open-source program.
3. The source code of a mature project is released to the public.
4. A well-established open-source project can be forked by an interested outside party.

Eric Raymond observed in his essay The Cathedral and the Bazaar that announcing the intent for a project is usually inferior to releasing a working project to the public.

It's a common mistake to start a project when contributing to an existing similar project would be more effective (NIH syndrome). To start a successful project it is very important to investigate what's already there. The process starts with a choice between the adopting of an existing project, and the starting of a new project. If a new project is started, the process goes to the Initiation phase. If an existing project is adopted, the process goes directly to the Execution phase.

UNIT – I

Topic:-Risk Assessment

Unit-01/Lecture-08

Risk Assessment:-[RGPV/JUNE-2012(10),JUNE-2014(5)]

The goal of risk assessment is to prioritize the risks so that attention and resources can be focused on the more risky items. Risk identification is the first step in risk assessment, which identifies all the different risks for a particular project. These risks are project-dependent and identifying them is an exercise in envisioning what can go wrong. Methods that can aid risk identification include checklists of possible risks, surveys, meetings and brainstorming, and reviews of plans, processes, and work products.

Checklists of frequently occurring risks are probably the most common tool for risk identification—most organizations prepare a list of commonly occurring risks for projects, prepared from a survey of previous projects. Such a list can form the starting point for identifying risks for the current project.

Based on surveys of experienced project managers, Boehm [11] has produced a list of the top 10 risk items likely to compromise the success of a software project. Figure shows some of these risks along with the techniques preferred by management for managing these risks. Top risks in a commercial software organization can be found in

	Risk Item	Risk Management Techniques
1	Personnel Shortfalls	Staffing with top talent; Job matching; Team building; Key personnel agreements; Training; Prescheduling key people
2	Unrealistic Schedules and Budgets	Detailed cost and schedule estimation; Design to cost; Incremental development; Software reuse; Requirements scrubbing
3	Developing the Wrong Software Functions	Organization analysis; Machine analysis; User surveys; Prototyping; Early user's manuals
4	Developing the Wrong User Interface	Prototyping; Scenarios; Task analysis; User characterization
5	Gold Plating	Requirements scrubbing; Prototyping; Cost benefit analysis; Design to cost

Figure 4.3: Top risk items and techniques for managing them

The top-ranked risk item is personnel shortfalls. This involves just having fewer people than necessary or not having people with specific skills that a project might require. Some of the ways to manage this risk are to get the top talent possible and to match the needs of the project with the skills of the available personnel. Adequate training, along with

having some key personnel for critical areas of the project, will also reduce this risk.

The second item, unrealistic schedules and budgets, happens very frequently due to business and other reasons. It is very common that high-level management imposes a schedule for a software project that is not based on the characteristics of the project and is unrealistic. Underestimation may also happen due to inexperience or optimism.

The next few items are related to requirements. Projects run the risk of developing the wrong software if the requirements analysis is not done properly and if development begins too early. Similarly, often improper user interface may be developed. This requires extensive rework of the user interface later or the software benefits are not obtained because users are reluctant to use it. Gold plating refers to adding features in the software that are only marginally useful. This adds unnecessary risk to the project because gold plating consumes resources and time with little return.

Risk identification merely identifies the undesirable events that might take place during the project, i.e., enumerates the "unforeseen" events that might occur. It does not specify the probabilities of these risks materializing nor the impact on the project if the risks indeed materialize. Hence, the next tasks are risk analysis and prioritization.

In risk analysis, the probability of occurrence of a risk has to be estimated, along with the loss that will occur if the risk does materialize. This is often done through discussion, using experience and understanding of the situation, though structured approaches also exist.

Once the probabilities of risks materializing and losses due to materialization of different risks have been analyzed, they can be prioritized. One approach for prioritization is through the concept of risk exposure (RE) [11], which is sometimes called risk impact. RE is defined by the relationship

RE = Prob(UO) * Loss(UO),

where Prob(UO) is the probability of the risk materializing (i.e., undesirable outcome) and Loss(UO) is the total loss incurred due to the unsatisfactory outcome. The loss is not only the direct financial loss that might be incurred but also any loss in terms of credibility, future business, and loss of property or life. The RE is the expected value of the loss due to a particular risk. For risk prioritization using RE is, the higher the RE, the higher the priority of the risk item.

Software Risk Management:
Since there could be various risks associated with the software development projects, the key to identify and manage those risks is to know about the concepts of software risk management. Many concepts about software risk management could be identified but the most important are risk index, risk analysis, and risk assessment-

1.Risk Index: Generally risks are categorized into two factors namely impact of risk events and probability of occurrence. Risk index is the multiplication of impact and probability of occurrence. Risk index can be characterized as high, medium, or low depending upon the product of impact and occurrence. Risk index is very important and necessary for prioritization of risk.

2. Risk Analysis: There are quite different types of risk analysis that can be used. Basically, risk analysis is used to identify the high risk elements of a project in software engineering. Also, it provides ways of detailing the impact of risk mitigation strategies. Risk analysis has also been found to be most important in the software design phase to evaluate criticality of the system, where risks are analyzed and necessary counter measures are introduced.

The main purpose of risk analysis is to understand risks in better ways and to verify and correct attributes. A successful risk analysis includes important elements like problem definition, problem formulation, data collection.

3. Risk Assessment: Risk assessment is another important case that integrates risk management and risk analysis. There are many risk assessment methodologies that focus on different types of risks. Risk assessment requires correct explanations of the target system and all security. It is important that a risk referent levels like performance, cost, support and schedule must be defined properly for risk assessment to be useful.

Risk Classification:

The key purpose of classifying risk is to get a collective viewpoint on a group of factors. These are the types of factors which will help project managers to identify the group that contributes the maximum risk. A best and most scientific way of approaching risks is to classify them based on risk attributes. Risk classification is considered as an economical way of analyzing risks and their causes by grouping similar risks together into classes. Software risks could be classified as internal or external. Those risks that come from risk factors within the organization are called internal risks whereas the external risks come from out of the organization and are difficult to control. Internal risks are project risks, process risks, and product risks. External risks are generally business with the vendor, technical risks, customers' satisfaction, political stability and so on. In general, there are many risks in the software engineering which is very difficult or impossible to identify all of them. Some of most important risks in software engineering project are categorized as software requirement risks, software cost risks, software scheduling risk, software quality risks, and software business risks. These risks are explained detail below –

1. SOFTWARE REQUIREMENT RISKS

- Lack of analysis for change of requirements.
- Change extension of requirements
- Lack of report for requirements
- Poor definition of requirements

- Ambiguity of requirements
- Change of requirements
- Inadequate of requirements
- Impossible requirements
- Invalid requirements

2.SOFTWARE COST RISKS

- Lack of good estimation in projects
- Unrealistic schedule
- The hardware does not work well
- Human errors
- Lack of testing
- Lack of monitoring
- Complexity of architecture
- Large size of architecture
- Extension of requirements change
- The tools does not work well
- Personnel change, Management change, technology change, and environment change
- Lack of reassessment of management cycle

3.SOFTWARE SCHEDULING RISKS

- Inadequate budget
- Change of requirements and extension of requirements
- Human errors
- Inadequate knowledge about tools and techniques
- Long-term training for personnel
- Lack of employment of manager experience
- Lack of enough skill
- Lack of good estimation in projects

4.SOFTWARE QUALITY RISKS

- Inadequate documentation
- Lack of project standard
- Lack of design documentation
- Inadequate budget
- Human errors
- Unrealistic schedule
- Extension of requirements change
- Poor definition of requirements
- Lack of enough skill
- Lack of testing and good estimation in projects

- Inadequate knowledge about techniques, programming language, tools, and so on

Strategies for Risk Management:

During the software development process various strategies for risk management could be identified and defined according to the amount of risk influence. Based upon the amount of risk influence in software development project, risk strategies could be divided into three classes namely careful, typical, and flexible, careful risk management strategy is projected for new and inexperienced organizations whose software development projects are connected with new and unproven technology; typical risk management strategy is well-defined as a support for mature organizations with experience in software development projects and used technologies, but whose projects carry a decent number of risks; and flexible risk management strategy is involved in experienced software development organizations whose software development projects are officially defined and based on proven technologies.

S.NO	RGPV QUESTIONS	Year	Marks
Q.1	What do you mean by risk assessment? Explain briefly	JUNE 2012	10

UNIT – I

Topic:- Difference between user requirements and specifications

Unit-01/Lecture-09

Difference between user requirements and specifications

This is a blog post I wrote for the intranet at my previous workplace to help influence a change in thinking in what constitutes “requirements”. Often, business will hand down “It must look like this and must work like this” which significantly constrains the design and is based on a lot of assumptions, competitors’ products and previous requirements. Plus it short-circuits any knowledge of technology, trends and patterns that the design and development team could have contributed.

USER REQUIREMENTS should detail what is needed by users; they should define the problem, the space within which a solution can be designed.

UI specifications detail a particular solution that can be implemented to meet the identified requirements.

There should be a clear and traceable relationship between every element of a design and the requirements.

The requirements and the specification can be in the same document but it should be clear that the requirements are defined and accepted first and that they are fixed whereas the solution specified is just one way that the requirements can be met.

Elements of a solution that cannot be traced back to a requirement indicate incomplete requirements or surplus design.

The following words should typically not be used when discussing user requirements as they refer to a user interface solution and confuse the purpose of a requirements document, preempt solutions and constrain the design, possibly excluding the optimal solution:

button, screen, panel, drop-down, list, scroll, click, tap, calendar, flip out, fold, animated, swipe, heading, label, line, grid, table, row, column, select, drag and drop, window, resize, right click, menu, sort, map, link, widget, box, icon, see, hear, touch …

This is a confused user requirement:

“A user can sort the table by any of the columns”

Rather, the user requirement might be:

“A user can opt to view the tabular data in ascending or descending order on any element of the data”

And the matching UI specification might be written (or committed straight to wireframes) as:

“In every cell of the header row will be two controls visible at all times, one for sorting the relevant column in ascending order and one for descending order”

REFERENCCE

BOOK	AUTHOR	PRIORITY
Software Engineering	P,S. Pressman	1
Software Engineering	Pankaj jalote	2