Process Models: Perspective Process Models (Continue)

LECTURE # 8

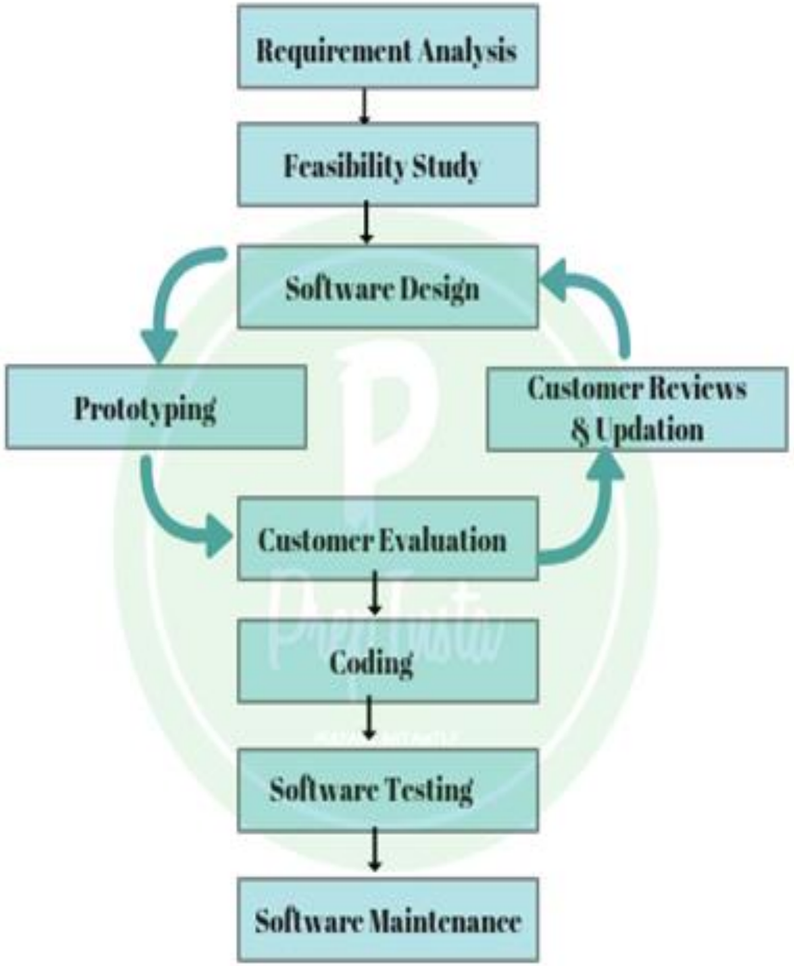
Lecture by Engr. Sidra

Prototype Model

- Prototype is a working model of software with some limited functionality.
- The basic idea in **Prototype model** is that instead of freezing the requirements before a design or coding can proceed, a throwaway prototype is built to understand the requirements.
- Prototype is developed based on the currently known requirements.
- By using prototype, the client can get an "actual feel" of the system, since the interactions with prototype can enable the client to better understand the requirements of the desired system.
- It does not always hold the exact logic used in the actual software application and is an extra effort to be considered under effort estimation
- Prototyping is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determining the requirements.

Prototype Model

- Software prototyping is becoming very popular as a software development model, as it enables to understand customer requirements at an early stage of development.
- It helps get valuable feedback from the customer and helps software designers and developers understand about what exactly is expected from the product under development
- Stepwise approach to design prototype
  - Basic Requirement Identification
  - Developing the initial Prototype
  - Review of the Prototype
  - Revise and Enhance the Prototype



Lecture by Engr. Sidra

Prototype Model

- **A Horizontal prototype** displays the user interface for the product and gives a broader view of the entire system, without concentrating on internal functions.
- **Horizontal prototypes** are used to get more information on the user interface level and the business requirements. It can even be presented in the sales demos to get business in the market
- A **Vertical prototype** on the other side is a detailed elaboration of a specific function or a sub system in the product.
- **Vertical prototypes** are technical in nature and are used to get details of the exact functioning of the sub systems. For example, database requirements, interaction and data processing loads in a given sub system.

Prototyping - Application

- Prototype model should be used when the desired system needs to have a lot of interaction with the end users.
- Prototyping ensures that the end users constantly work with the system and provide a feedback which is incorporated in the prototype to result in a useable system.
- When detailed information related to input and output requirements of the system is not available
- They are excellent for designing good human computer interface systems.

Advantages of Prototype model:

- Users are actively involved in the development
- Since in this methodology a working model of the system is provided, the users get a better understanding of the system being developed.
- Errors can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily
- Confusing or difficult functions can be identified
- Reduces time and cost.

Lecture by Engr. Sidra

Disadvantages of Prototype model

- Leads to implementing and then repairing way of building systems.
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
- Risk of insufficient requirement analysis owing to too much dependency on the prototype.
- Users may get confused in the prototypes and actual systems.
- Developers may try to reuse the existing prototypes to build the actual system, even when it is not technically feasible.
- The effort invested in building prototypes may be too much if it is not monitored properly.

Summary

- A generic process model for software engineering encompasses a set of framework and umbrella activities, actions, and work tasks.
- Each of a variety of process models can be described by a different process flow
- Process patterns can be used to solve common problems that are encountered as part of the software process.
- Prescriptive process models have been applied for many years for software development. Each of these models suggests a some- what different process flow, but all perform the same set of generic framework activities: communication, planning, modeling, construction, and deployment.

Summary

- Sequential process models, such as the waterfall and V models, are the oldest software engineering paradigms. They suggest a linear process flow that is often in- consistent with modern, however, have applicability in situations where requirements are well defined and stable.
- Incremental process models are iterative in nature and produce working versions of software quite rapidly and are designed to accommodate change.
- Evolutionary models, such as prototyping and the spiral model, produce incremental work products quickly. These models can be adopted to apply across all software engineering activities—from concept development to long-term system maintenance.

Process Models: Specialized Process Models

LECTURE # 8

Lecture by Engr. Sidra

Objectives

- The objectives of this lecture is to
  - Understand the Specialized Process Models
  - Their strengths and weaknesses

Specialized Process model

- Special process models take many features from one or more conventional models.
- However these special models tend to be applied when a narrowly defined software engineering approach is chosen.
- Types in Specialized process models:
  - 1. Component based development (Promotes reusable components)
  - 2. The formal methods model (Mathematical formal methods are backbone here)
  - 3. Aspect oriented software development (Uses crosscutting technology)
  - Unified Process (use-case driven, architecture centric)

Component Based Development

- Software Reuse:
  - In most engineering disciplines, systems are designed by composition (building system out of components that have been used in other systems)
  - Software engineering has focused on custom development of components
  - To achieve better software quality, more quickly, at lower costs, software engineers are beginning to adopt systematic reuse as a design process

Component Based Development

- **Benefits of Reuse**
  - Increased Reliability
  - Reduced Process Risk
  - Effective Use of Specialists
  - Standards Compliance
  - Accelerated Development

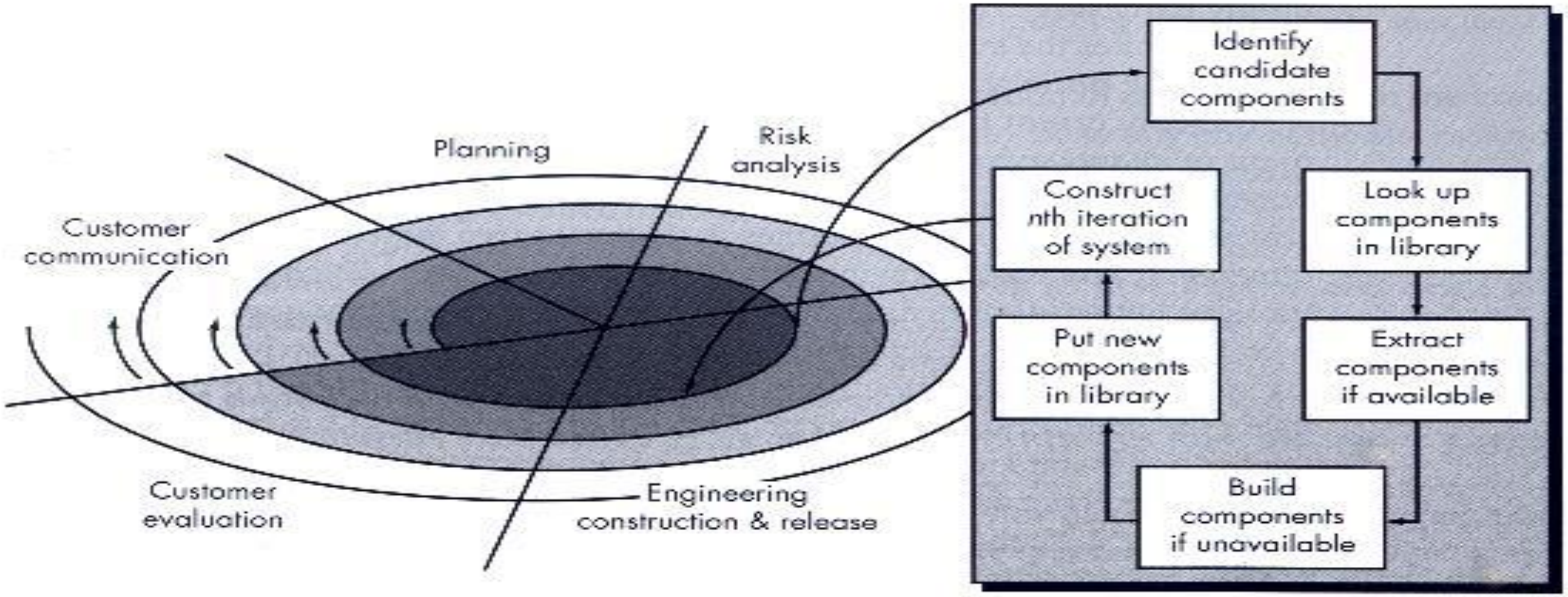
Component Based Development (CBD)

- Component-based software engineering is the idea of building software from established software components, as opposed to building the software from the scratch.
- Commercial off-the-shelf (COTS) software components, developed by vendors who offer them as products, provide targeted functionality with well-defined interfaces that enable the component to be integrated into the software that is to be built.
- Components interact through well-defined interfaces.

Component Based Development (CBD)

- The component based development model incorporates many of the characteristics of the spiral model.
- It is evolutionary in nature, demanding an iterative approach to the creation of software.
- However, the model focuses on prepackaged software components. It promotes software reusability.
- Component integration is relatively easy, the main focus is on maintenance.

Component Based Development (CBD)


Planning
Risk analysis
Customer communication
Customer evaluation
Engineering construction & release
Identify candidate components
Look up components in library
Extract components if available
Build components if unavailable
Put new components in library
Construct nth iteration of system

Lecture by Engr. Sidra

Component Based Development (CBD)

- Modeling and construction activities begin with the identification of candidate components. Candidate components can be designed as either conventional software modules or object oriented packages.
- Component based development has the following steps:
  - 1. Available component based products are researched and evaluated for the application domain.
  - 2. Component integration issues are considered.
  - 3. A software architecture is designed to accommodate the components.
  - 4. Components are integrated into the architecture.
  - 5. Comprehensive testing is conducted to ensure proper functionality.