Session – 1

SOFTWARE

K L University

u/s 3 of UGC Act. 1956

Koneru Lakshmaiah Education Foundation

INTRODUCTION

What is Software

Software is:

instructions (computer programs) that when executed provide desired features, function, and performance; .

Software is a set of instructions, data or programs used to operate computers and execute specific tasks

Characteristics of S/W

1. *Software is developed or engineered, it is not manufactured in the classical sense.*

- *Software is logical and hardware is physical entity. Both activities require the construction of a "product," but the approaches are different*

2. *Software doesn't "wear out."* but it *deteriorates*

Hardware effected by heat ,dust etc., ,software does not effected by environmental conditions

3. *Although the industry is moving toward component-based construction, most software continues to be custom-built.*

A software component should be designed and implemented so that it can be reused in many different programs

Failure curves for hardware

Failure curves for software

Wear vs. Deterioration

Nature of Software

Software is a **Product**

- *Transforms* information- produces, manages, acquires, modifies, displays or transmits information
- Delivers computing potential of hardware and networks

Software is a **vehicle** for delivering a product

- Controls other programs (operating system)
- Effects communications (networking software)
- Helps build other software (software tools & environments)

Software Application Domains

K L University
u/s 3 of UGC Act. 1956
Koneru Lakshmaiah Education Foundation

Software Application Domains- 7

- System software
- Application software
- Engineering/scientific software
- Embedded software
- Product-line software
- WebApps (Web applications)
- AI software

- **System software—a collection of programs written to service other programs.**

  Some system software (e.g., compilers)

- **Application software—stand-alone programs that solve a specific business** need.

(e.g., Ticket booking from irctc,Passport application ,real-time manufacturing process control).

- **Engineering/scientific software—has been characterized by "number** crunching" algorithms

(e.g., Computer-aided design, system simulation,

and other interactive applications)

- **Embedded software—resides within a product or system and is used to** implement and control features

e.g., key pad control for a microwave oven)

- **Product-line software—designed to provide a specific capability for use by** many different customers. (e.g., word processing, spreadsheets
- **Web applications—called "WebApps," this network-centric software** (e.g., True caller.. etc)
- **Artificial intelligence software—makes use of nonnumerical algorithms to** solve complex problems (e.g., game playing etc)

Software - New Challenges

- Open world computing—pervasive, distributed computing
- Ubiquitous computing—wireless networks
- Net sourcing—the Web as a computing engine-Need target marketing
- Open source—”free” source code open to the computing community

Legacy Software

- older programs—often referred to as *legacy software*
- They were developed decades ago and have been continually modified to meet changes in business requirements
- one additional characteristic that is present in legacy software—*poor quality*

- a legacy system must be reengineered

legacy systems often evolve for one or more of the following reasons:

- The software must be adapted to meet the needs of new computing environments or technology.
- The software must be enhanced to implement new business requirements.
- The software must be extended to make it interoperable with other more modern systems or databases.
- The software must be re-architected to make it viable within a network environment.

Unique Nature of WebApps

- **Network intensiveness.** A WebApp resides on a network and must serve the needs of a diverse community of clients.

- **Concurrency.** A large number of users may access the WebApp at one time.

- **Unpredictable load.** The number of users of the WebApp may vary by orders of magnitude from day to day.

- **Performance.** If a WebApp user must wait too long (for access, for server-side processing, for client-side formatting and display), he or she may decide to go elsewhere.

- **Availability.** Although expectation of 100 percent availability is unreasonable, users of popular WebApps often demand access on a “24/7/365” basis.

Unique Nature of WebApps

- **Data driven.** The primary function of many WebApps is to use hypermedia to present text, graphics, audio, and video content to the end-user.
- **Content sensitive.** The quality and aesthetic nature of content remains an important determinant of the quality of a WebApp.
- **Continuous evolution.** Unlike conventional application software that evolves over a series of planned, chronologically-spaced releases, Web applications evolve continuously.
- **Immediacy.** Although *immediacy*—the compelling need to get software to market quickly—is a characteristic of many application domains, WebApps often exhibit a time to market that can be a matter of a few days or weeks.
- **Security.** Because WebApps are available via network access, it is difficult, if not impossible, to limit the population of end-users who may access the application..

End of session - 1

Questions

1. Define Software.
2. What is Bathtub curve.
3. What are Characteristics of S/W.
4. What are different Software Application Domains.
5. Advantages of Software

Session – 2

Software Engineering

K L University

u/s 3 of UGC Act. 1956

Koneru Lakshmaiah Education Foundation

Software Engineering

- Some realities:
  - *a concerted effort should be made to understand the problem before a software solution is developed*
  - *design becomes a pivotal activity*
  - *software should exhibit high quality*
  - *software should be maintainable*
- The seminal definition:
  - *[Software engineering is] the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.*

Software Engineering

- The IEEE definition:
  - *Software Engineering:*

    The application of a
    Systematic,
    Disciplined,
    Quantifiable approach
    to the development, operation, and maintenance of software;

    that is, the application of engineering to software.

A Layered Technology

Software Engineering

A Layered Technology

Quality: Organization is commitment to quality

Process: Manages the control of software projects
Ensures quality, establishes milestones, manages changes

Methods: Provide technical ways for building software i.e;
Communication, requirement analysis, design modeling,
program construction , testing and support

Tools: Provide automated or semi automated support for the
process & methods

The Software Process

A ***Process*** is collection of activities, actions, and tasks that are performed when some work product is to be created

- ***Activity-***Strives to achieve a broad objective

  (eg: communication with stakeholders)

- ***Actions-*** encompasses a set of tasks that produce a major work

  product ***(eg:*** architectural design***)***

- ***Task-***focusses on small, but well-defined objective that

  produces a tangible outcome ***(eg:*** conduct a unit test***)***

Process framework Activities

1. **Communication**
2. **Planning**
3. **Modeling**
   - Analysis of requirements
   - Design
4. **Construction**
   - Code generation
   - Testing
5. **Deployment**
   - Delivery
   - Feedback

Process framework Activities(cont.)

- **Communication:** communicate with customer to understand objectives and gather requirements
- **Planning**:
- **Modeling:**
- **Construction**:
- **Deployment:**

Process framework Activities(cont.)

- **Communication:**
- **Planning**: creates a "map" that defines the work by describing tasks, risks and resources, work products and work schedule.
- **Modeling:**
- **Construction**:
- **Deployment:**

Process framework Activities(cont.)

- **Communication:**
- **Planning**:
- **Modeling:** Create a "sketch", what it looks like architecturally, how the essential parts fit together and other characteristics.
- **Construction**:
- **Deployment:**

Process framework Activities(cont.)

- **Communication:**
- **Planning**:
- **Modeling:**
- **Construction:** code generation and the testing.
- **Deployment:**

Process framework Activities(cont.)

- **Communication:**
- **Planning**:
- **Modeling:**
- **Construction**:
- **Deployment:** Delivered to the customer who evaluates the products & provides feedback based on the evaluation.

Process framework Activities(cont.)

- These five framework activities can be used to all software development, regardless of the application domain, size of the project, complexity of the efforts etc.
  - though the details will be different in each case.

- For many software projects, these framework activities are applied **iteratively** as a project progresses. Each iteration produces a software increment that provides a subset of overall software features and functionality.

Umbrella Activities

- Complete the five process framework activities and help team manage and control progress, quality, change, and risk.

Software project tracking and control:

- **Risk management**:
- **Software quality assurance**:
- **Technical reviews**:
- **Measurement**:
- **Software configuration management**:
- **Reusability management**:
- **Work product preparation and production:**

Umbrella Activities

- **Software project tracking & control**: assess progress against the plan and take actions to maintain the schedule.
- **Risk management**:
- **Software quality assurance**:
- **Technical reviews**:
- **Measurement**:
- **Software configuration management**:
- **Reusability management**:
- **Work product preparation and production:**

Umbrella Activities

- **Software project tracking & control**:
- **Risk management**: assesses risks that may affect the outcome and quality.
- **Software quality assurance**:
- **Technical reviews**:
- **Measurement**:
- **Software configuration management**:
- **Reusability management**:
- **Work product preparation and production:**

Umbrella Activities

- **Software project tracking & control**:
- **Risk management**:
- **Software quality assurance**: defines and conduct activities to ensure quality.
- **Technical reviews**:
- **Measurement**:
- **Software configuration management**:
- **Reusability management**:
- **Work product preparation and production:**

Umbrella Activities

- **Software project tracking & control**:
- **Risk management**:
- **Software quality assurance**:
- **Technical reviews**: assesses work products to uncover and remove errors before going to the next activity.
- **Measurement**:
- **Software configuration management**:
- **Reusability management**:
- **Work product preparation and production:**

Umbrella Activities

- **Software project tracking & control**:
- **Risk management**:
- **Software quality assurance**:
- **Technical reviews**:
- **Measurement**: define and collects process, project, and product measures to ensure stakeholder's needs are met.
- **Software configuration management**:
- **Reusability management**:
- **Work product preparation and production:**

Umbrella Activities

- **Software project tracking & control**:
- **Risk management**:
- **Software quality assurance**:
- **Technical reviews**:
- **Measurement**:
- **Software configuration management**: manage the effects of change throughout the software process.
- **Reusability management**:
- **Work product preparation and production:**

Umbrella Activities

- **Software project tracking & control**:
- **Risk management**:
- **Software quality assurance**:
- **Technical reviews**:
- **Measurement**:
- **Software configuration management**:
- **Reusability management**: defines criteria for work product reuse and establishes mechanism to achieve reusable components.
- **Work product preparation and production:**

Umbrella Activities

- **Software project tracking & control**:
- **Risk management**:
- **Software quality assurance**:
- **Technical reviews**:
- **Measurement**:
- **Software configuration management**:
- **Reusability management**:
- **Work product preparation and production:** create work products such as models, documents, logs, forms and lists.

Software Engineering Practice

Software Engineering Practice

The Essence of Practice

- How does the practice of software engineering fit in the process activities mentioned above? Namely,
  - **communication,**
  - **planning,**
  - **modeling,**
  - **construction**
  - **deployment**.
- The essence of problem solving is outlined in 4 points:

1. *Understand the problem* (communication and analysis).
2. *Plan a solution* (modeling and software design).
3. *Carry out the plan* (code generation).
4. *Examine the result for accuracy* (testing and quality assurance).

Understand the Problem

- ***Who has a stake in the solution to the problem?*** That is, who are the stakeholders?
- ***What are the unknowns?*** What data, functions, and features are required to properly solve the problem?
- ***Can the problem be compartmentalized?*** Is it possible to represent smaller problems that may be easier to understand?
- ***Can the problem be represented graphically?*** Can an analysis model be created?

Plan the Solution

- ***Have you seen similar problems before?*** Are there patterns that are recognizable in a potential solution? Is there existing software that implements the data, functions, and features that are required?
- ***Has a similar problem been solved?*** If so, are elements of the solution reusable?
- ***Can subproblems be defined?*** If so, are solutions readily apparent for the subproblems?
- ***Can you represent a solution in a manner that leads to effective implementation?*** Can a design model be created?

Carry Out the Plan

- ***Does the solution conform to the plan?*** Is source code traceable to the design model?
- ***Is each component part of the solution provably correct?*** Has the design and code been reviewed, or better, have correctness proofs been applied to algorithm?

Examine the Result

- ***Is it possible to test each component part of the solution?*** Has a reasonable testing strategy been implemented?
- ***Does the solution produce results that conform to the data, functions, and features that are required?*** Has the software been validated against all stakeholder requirements?

Hooker’s General Principles

1: *The Reason It All Exists*

2: *KISS (Keep It Simple, Stupid!)*

3: *Maintain the Vision*

4: *What You Produce, Others Will Consume*

5: *Be Open to the Future*

6: *Plan Ahead for Reuse*

7*: Think!*

Software Myths

1.Management Myths

- “We already have a book of standards and procedures for building software. It does provide my people with everything they need to know …”
- “If my project is behind the schedule, I always can add more programmers to it and catch up …”

  (a.k.a. “**The Mongolian Horde concept**”)
- “If I decide to outsource the software project to a third party, I can just relax: Let them build it, and I will just pocket my profits …”

2. Customer Myths

- “A general statement of objectives is sufficient to begin writing programs - we can fill in the details later …”

- “Project requirements continually change but this change can easily be accommodated because software is flexible …”

3. Practitioner's Myths

- "Let's start coding ASAP, because once we write the program and get it to work, our job is done …"
- "Until I get the program running, I have no way of assessing its quality …"
- "The only deliverable work product for a successful project is the working program …"
- "Software engineering is baloney(nonsense.). It makes us create tons of paperwork, only to slow us down …"

End of Session - 2

Questions:

1. Define Software Engineering.
2. What is the need of Umbrella Activities.
3. What are different Layers of Software Engineering.
4. What are Process framework Activities.
5. What are different activities comes under Modeling face.
6. Need of Software quality assurance in software development process.
7. Define Myth ?
8. How many types of Myths available.
9. What are myths comes under Management Myths.
10. What are myths comes under Customer Myths.
11. What are myths comes under Practitioner's Myths.

Session -3

A Generic Process Model

K L University
u/s 3 of UGC Act. 1956
Koneru Lakshmaiah Education Foundation

- a process was defined as a collection of work activities, actions, and tasks that are performed when some work product is to be created

- each framework activity is populated by a set of software engineering actions.

- Each software engineering action is defined by a *task set that*
- *identifies the work* tasks that are to be completed,
- the work products that will be produced,
- the quality assurance points that will be required,
- and the milestones that will be used to indicate progress.

A Generic Process Model

Software process

Process Flow: How the framework activities, actions and tasks occur within each framework activity

A *linear process flow executes each of the five framework activities in sequence,* beginning with communication and culminating with deployment

An *iterative process flow repeats one or more of the activities before proceeding to the* next.

An *evolutionary process flow executes the activities in a "circular"* manner. Each circuit through the five activities leads to a more complete version of the software

A *parallel process flow executes one or* more activities in parallel with other activities (e.g., modeling for one aspect of the software might be executed in parallel with construction of another aspect of the software).

(a) Linear process flow

(b) Iterative process flow

(c) Evolutionary process flow

(d) Parallel process flow

Prescriptive Process models

K L University
u/s 3 of UGC Act. 1956
Koneru Lakshmaiah Education Foundation

Prescriptive Process models

- Waterfall Model.
- Incremental Process Model.
- Evolutionary Process Model.
- Concurrent model

Prescriptive Process models

Waterfall Model

- Oldest software lifecycle model & best understood by upper management
- Used when requirements are well understood and risk is low
- Work flow is in a linear fashion (i.e., sequential)
- Used often with well-defined adaptations or enhancements to current software

Communication
Project initiation
Requirement gathering
Planning
Estimating
Scheduling
Tracking
Modeling
Analysis
Design
Construction
Code
Test
Deployment
Delivery
Support
Feedback

Waterfall Model

Useful

- Where requirements are fixed and is to proceed to completion in a linear manner.

Cons in Water fall model

- Real projects rarely follow the sequential flow that the model proposes
- It is often difficult for the customer to state all requirements explicitly
- The customer must have patience. A working version of the program(s) will not be available until late in the project time span

The
desi
acc

Engineering: A Practitioner's Approach, 7/e

- The V-model provides a way of visualizing how verification and validation actions are applied to earlier engineering work

- V- model means Verification and Validation model. Just like the **waterfall model**, the V-Shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins

- **Advantages of V-model:**

- Simple and easy to use.

- Testing activities like planning, **test designing** happens well before coding. This saves a lot of time. Hence higher chance of success over the waterfall model.

- Defects are found at early stage.

- Works well for small projects where requirements are easily understood.

- **Disadvantages of V-model:**

- Very rigid and least flexible.

- Software is developed during the implementation phase, so no early prototypes of the software are produced.

- If any changes happen in midway, then the test documents along with requirement documents has to be updated.

- **When to use the V-model:**
- The V-shaped model should be used for small to medium sized projects where requirements are clearly defined and fixed.
- The V-Shaped model should be chosen when ample technical resources are available with needed technical expertise.

Prescriptive Process models

Incremental Process Model

- Combines elements of linear and parallel process flows.
- It delivers a series of releases, called increments that provide progressively more functionality for the customer as each is delivered
- The first increment is often a core product.
- The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional feature and functionality.
- It focuses on the delivery of an operational product with each increment.

Incremental Process Model

- It is useful when staffing is unavailable for a complete implementation.
- Increments can be planned to manage technical risks.

For example, word-processing software developed using the incremental paradigm might deliver basic file management, editing, and document production functions in the first increment; more sophisticated editing and document production capabilities in the second increment; spelling and grammar checking in the third increment; and advanced page layout capability in the fourth increment

- **Advantages of Incremental model:**
- Generates working software quickly and early during the software life cycle.
- This model is more flexible – less costly to change scope and requirements.
- It is easier to test and debug during a smaller iteration.
- In this model customer can respond to each built.
- Lowers initial delivery cost.
- Easier to manage risk because risky pieces are identified and handled during it’s iteration.

- **Disadvantages of Incremental model:**
- Needs good planning and design.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than **waterfall**.

- **When to use the Incremental model:**
- This model can be used when the requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some details can evolve with time.
- There is a need to get a product to the market early.
- A new technology is being used

Evolutionary Process Model

- Software, like Complex systems evolve over a period of time

  -Business and product requirements often change as development proceeds.

  - tight market deadlines make completion of a comprehensive software product impossible, but a limited version must be introduced to meet competitive or business pressure

  - a set of core product or system requirements is well understood, but the details of product or system extensions have yet to be defined.

- Evolutionary models are iterative.
- Evolutionary Process Model produce an increasingly more complete version of the software with each iteration.
- Specification, development and validation are interleaved.

Prescriptive Process models

<u>Evolutionary Process Model</u>

- Prototyping
- Spiral Model

Prototype Model is a software development life cycle model which is used when the customer is not known completely about how the end product should be and its requirements. So in this model, a prototype of the end product is first developed by the developers and then tested and changes were made as per customer feedback until the customer is satisfied with the prototype.

Evolutionary Process Model

1. Prototyping

- It assists you and stakeholders to better understand what is to built when requirements are fuzzy.
- Prototyping paradigm

  -begins with communication.

  - planned quickly and modelling occurs

  - quick design focuses on a representation of those aspects of the software that will be visible to end users.
- It serves as a mechanism for identifying software requirements.
- a first or preliminary version of a device or vehicle from which other forms are developed

Prototyping Model(Diagram)

- **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

- **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.
- Incomplete application may cause application not to be used as the full system was designed
- Incomplete or inadequate problem analysis.
- **When to use Prototype model:**
- Prototype model should be used when the desired system needs to have a lot of interaction with the end users.
- Typically, online systems, web interfaces have a very high amount of interaction with end users, are best suited for Prototype model. It might take a while for a system to be built that allows ease of use and needs minimal training for the end user.

Evolutionary Process Model

2. Spiral development

- Originally proposed by Barry Boehm.
- It couples the iterative nature of prototyping with the controlled and systematic aspects of water fall model.
- Process is represented as a spiral rather than as a sequence of activities with backtracking.
- Each loop in the spiral represents a phase in the process.
- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required.
- Risks are explicitly assessed and resolved throughout the process.

The process of development is fast.
There is control towards all phases of development.
Customer feedback is taken into consideration and changes implemented as soon as possible.
Many and more features are added systematically.
Most ideal for large and risky projects.

Spiral model

- It is a realistic approach to the development of large-scale systems and software.
- The software evolves as the process progresses, the developer and customer better understand and react at each evolutionary level.
- It uses prototyping as a risk reduction mechanism.
- It demands considerable risk assessment expertise and realise on this expertise for success.

Spiral Model (Diagram)

- **Advantages of Spiral model:**
- High amount of risk analysis hence, avoidance of Risk is enhanced.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the **software life cycle**.

- **Disadvantages of Spiral model:**
- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project’s success is highly dependent on the risk analysis phase.
- Doesn’t work well for smaller projects.
-

- **When to use Spiral model:**
- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex

Evolutionary development

- Problems
  - Do not establish the max. speed of the evolution.
  - Systems are often poorly structured;
  - Special skills (e.g. in languages for rapid prototyping) may be required.
  - Project management and estimation technique do not fit completely.

- Applicability
  - For small or medium-size interactive systems;
  - For parts of large systems (e.g. the user interface);
  - For short-lifetime systems.

Concurrent Model

- The concurrent development model sometimes called Concurrent Engineering.
- It allows team to represent iterative and concurrent elements of any of the process models.
- All software engineering activities exist concurrently but reside in different states.
- Concurrent modeling defines a series of events that will trigger transitions from state to state for each of the activities.
- Concurrent modeling is applicable to all types of software development and provide an accurate picture of the current state of a project.

Concurrent models are those **models** within which the various activities of software development happen at the same time, for faster development and a better outcome. The **concurrent model** is also referred to as a parallel working **model**

Advantages of the concurrent development model

This model is applicable to all types of software development processes.

It is easy for understanding and use.

It gives immediate feedback from testing.

It provides an accurate picture of the current state of a project.

Disadvantages of the concurrent development model

It needs better communication between the team members. This may not be achieved all the time.

It requires to remember the status of the different activities.

One element of the concurrent process model

End of Session - 3

Questions:

1. What Classic life cycle model.
2. Disadvantages of Incremental Process Model.
3. Main drawback of Prototyping Model.
4. Speciality in Spiral Model.
5. Need of Process models in Software development.
6. Advantages of Spiral model.
7. Disadvantages of Spiral model.
8. Advantages of Evolutionary model.
9. Disadvantages of Evolutionary model.
10. Specialty of Concurrent model.
11. Difference between Waterfall model and Spiral model.

Session – 4

Specialized Process Models

K L University

u/s 3 of UGC Act. 1956

Koneru Lakshmaiah Education Foundation

Component-based Development Model

- The process to apply when reuse is a development objective
- It Consists of the following process steps
  - Available component-based products are researched and evaluated for the application domain in question
  - Component integration issues are considered
  - A software architecture is designed to accommodate the components
  - Components are integrated into the architecture
  - Comprehensive testing is conducted to ensure proper functionality

- Capitalizes on software reuse, which leads to documented savings in project cost and time

Formal Methods Model (Description)

- Encompasses a set of activities that leads to formal mathematical specification of computer software
- Enables a software engineer to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notation
- Ambiguity, incompleteness, and inconsistency can be discovered and corrected more easily through mathematical analysis
- Offers the promise of defect-free software
- Used often when building safety-critical systems

Formal Methods Model (Challenges)

- Development of formal methods is currently quite time-consuming and expensive
- Because few software developers have the necessary background to apply formal methods, extensive training is required
- It is difficult to use the models as a communication mechanism for technically unsophisticated customers

Aspect-Oriented Software Development

It provides a process and methodological approach for defining, specifying, designing, and constructing aspects

As modern computer based systems become more sophisticated and complex there are certain ***concerns by the customer***

- required properties or areas of technical interest.
- Span the entire architecture
- High-level properties of a system (e.g; security, fault tolerance)
- Other concern affect functions (e.g; the application of business rules)
- While others are systemic (e,g; task synchronization or memory management)

The Unified Process

K L University

u/s 3 of UGC Act. 1956

Koneru Lakshmaiah Education Foundation

Background

- During early 1990s JmaesRumbaugh, Grady Booch and Ivar Jacobson eventually worked together on a unified method, called the Unified Modelling Language (UML)
  - UML is a robust notation for the modelling and development of object-oriented systems
  - UML became an industry standard in 1997
  - However, UML does not provide the process framework, only the necessary technology for object-oriented development

- <u>Unified process</u> developed which is a framework for object-oriented software engineering using UML
  - Draws on the best features and characteristics of conventional software process models
  - Emphasizes the important role of software architecture
  - Consists of a process flow that is iterative and incremental, thereby providing an evolutionary feel

Background (continued)

- Consists of 5 phases:
  1. Inception
  2. Elaboration
  3. Construction
  4. Transition
  5. Production

Phases of the Unified Process

(1) - Inception Phase

- Encompasses both customer communication and planning activities of the generic process
- Business requirements for the software are identified
- A rough architecture for the system is proposed
- A plan is created for an incremental, iterative development
- Fundamental business requirements are described through preliminary use cases
  - A use case describes a sequence of actions that are performed by a user

(2) - Elaboration Phase

- Encompasses both the planning and modelling activities of the generic process
- Refines and expands the preliminary use cases
- Expands the architectural representation to include five views
  - Use-case model
  - Analysis model
  - Design model
  - Implementation model
  - Deployment model
- Often results in an executable architectural baseline that <u>represents a first cut executable system</u>
- The baseline demonstrates the viability of the architecture but <u>does not provide all features and functions required to use the system</u>

(3) - Construction Phase

- Encompasses the construction activity of the generic process
- Uses the architectural model from the elaboration phase as input
- Develops or acquires the software components that make each use -case operational
- Analysis and design models from the previous phase are completed to reflect the final version of the increment
- Use cases are used to derive a set of acceptance tests that are executed prior to the next phase

(4) - Transition Phase

- Encompasses the last part of the construction activity and the first part of the deployment activity of the generic process
- Software is given to end users for beta testing and user feedback reports on defects and necessary changes
- The software teams create necessary support documentation (user manuals, trouble-shooting guides, installation procedures)
- At the conclusion of this phase, the software increment becomes a usable software release

(5) - Production Phase

- Encompasses the last part of the deployment activity of the generic process
- On-going use of the software is monitored
- Support for the operating environment (infrastructure) is provided
- Defect reports and requests for changes are submitted and evaluated

Unified Process Work Products

- Work products are produced in each of the first four phases of the unified process
- In this course, we will concentrate on the analysis model and the design model work products
- Analysis model includes
  - Scenario-based model, class-based model, and behavioural model
- Design model includes
  - Component-level design, interface design, architectural design, and data/class design

End of session - 4

☺

Questions:

1. Main importance of Unified Process model.
2. Various Challenges in Formal Methods Model.
3. Difference between Inception and Elaboration.
4. Importance of Construction Phase in Unified process model.
5. Need and importance of UML in software development.

Session -5

Personal and Team Process models

Personal and Team Process models

- The software process is effective only if it is amenable to significant adaption to meet the needs of the project team.
- **Personal process model:** Individual create a process that best fits your needs, and at the same time, meets the broader needs of the team and the organization.
- **Team process model:** The Team itself create its own process, and at the same time meet the narrower needs of individuals and the broader needs of the organisation.

"Watts Humphrey" *argues that both models are achievable but require hard work, training and coordination.*

PSP shows Engineers how to

- Manage the quality of their prog
- Make commitments
- Improve estimating & planning
- reduce the defects.

Personal Software Process (PSP)

Watts Humphrey suggests that in order to change an ineffective personal process, an individual must move through four phases, each requiring training and careful instrumentation.

The PSP

- emphasizes personal measurement of the work product that is produced .
- emphasizes the resultant quality of the work product.
- Makes practitioner responsible for project planning (e.g; estimating and scheduling)
- Empowers the practitioner to control the quality of all software work products that are developed.

Personal Software Process (PSP) defines 5 framework activities

- **Planning.** This activity isolates requirements and develops both size and resource estimates. In addition, a defect estimate (the number of defects projected for the work) is made. All metrics are recorded on worksheets or templates. Finally, development tasks are identified and a project schedule is created.

- **High-level design.** An external specification is created for each component and a component design is created. Prototypes are built when uncertainty exists. All issues are recorded and tracked.

Personal Software Process (PSP)

- **High-level design review**. Formal verification methods are applied to uncover errors in the design. Metrics are maintained for all important tasks and work results.

- **Development.** The component level design is refined and reviewed. Code is generated, reviewed, compiled, and tested. Metrics are maintained for all important tasks and work results.

- **Postmortem.** Using measures and metrics collected the effectiveness of the process is determined. (Final O/P If this is a large amount of data it should be analyzed statistically.) Measures and metrics should provide guidance for modifying the process to improve its effectiveness.

Personal Software Process (PSP)

- PSP Stresses the need for each software engineer to identify errors early and as important, to understand the types of errors through a rigorous assessment activity performed on all work products you produce.

- However, PSP has not been widely adopted throughout the industry.
- PSP is intellectually challenging and demands a level of commitment (by practitioners and managers) that is not always possible to obtain.
- Training is relatively lengthy and training costs are high.
- The required level of measurement is culturally difficult for many software people.

Team Software Process (TSP)

The goal of TSP is to build a "self-directed" project team that organizes itself to produce high quality software. **Humphrey** defines the following objectives for TSP.

- Build self-directed teams that plan and track their work, establish goals, and own their processes and plans. These can be pure software teams or integrated product teams (IPT) of three to about 20 engineers.
- Show managers how to coach and motivate their teams and how to help them sustain peak performance.
- Accelerate software process improvement by making CMM Level 5 behavior normal and expected.
  - The Capability Maturity Model (CMM), a measure of the effectiveness of a software process, is discussed in Chapter 30.
- Provide improvement guidance to high-maturity organizations.
- Facilitate university teaching of industrial-grade team skills. **End of Session -5**

Additional topics for self learning:

- Product and Process

Questions:

1. **What is** PSP.
2. What is TSP.
3. Differences between PSP and TSP.
4. How PSP used for Quality improvement.
5. How TSP used for Quality improvement.

Session - 6

Reverse Engineering

- Forward Engineering
- In software development life cycle , we start from requirement phase ,analysis , design and implementation

- Reverse Engineering
- It is the process of creating of design document from source code and the requirements from design

S.N.	Forward engineering	Reverse engineering
1.	Applications are developed with given requirements	Information for the system are collected from the given application
2.	It's flow is model => System	It's flow is System => model
3.	Takes more time for development	Takes less time for development
4.	Prescriptive, Developers are told how to work.	Adaptive, Engineer must find out what actually the developer did
5.	Production is started with given requirements.	Production is started by taking existing product.
6.	It requires high proficiency skills	It may not require high proficiency skills
7.	For example:- Developing a new software from scratch	For example:- Cloning Facebook, Instagram, Paypal

Forward Engineering
Reverse Engineering
requirements
Analysis
Design
Implementation

It is the process followed as to find unknown and difficult information about a software

Why it is important

It is important in case when software lacks proper documentation , and it is highly unstructured

It is degraded through maintenance works

Purpose

Recovering information from existing code and intermediate documents

Program understanding at any level it is included in RE

- Reverse Engineering Goals

1)Recover Lost information

2)Cope with complexity

3)Detect side Effects

4)Synthesize higher abstraction

5)Facilitate reusability

Uses of reverse engineering

Program understanding

Re-documentation or document generation

Recovery of design details

Business rules implied on software

Understanding high level system description

Requirement Engineering activities

- Understanding the process
- Understanding the data
- Understanding user interfaces

LEVELS OF REVERSE ENGINEERING
Abstraction Level
High
Intermed-
-iary
Inter-
-mediary
Low
Specification
Design
Implementation
Specification Recovery
Design Recovery
Redocu-
mentation
Redoc-
-umentation
Redoc-
-umentation
Reverse Engineering

Questions:

1. Need of Reverse engineering.
2. How Forward Engineering work.
3. Different levels of reverse engineering

Session – 7

Understanding Requirements

Need to Understand Requirements (Quick Look)

- What is it?
- Who does it?
- Why is it important?
- What are the Steps?
- What is the work product?
- How do I ensure that I've done it right?

How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Types of Requirements

Broadly requirements can be categorized into

- Functional
- Non Functional

Requirement Engineering

Requirement Engineering is the process of defining, documenting and maintaining the requirements. It is a process of gathering and defining service provided by the system.

Stages in Requirements Engineering

- Elicitation
- Elaboration
- Negotiation
- Specification
- Validation
- Requirements management

Contd.,

- Inception—ask a set of questions that establish …
  - basic understanding of the problem
  - the people who want a solution
  - the nature of the solution that is desired, and
  - the effectiveness of preliminary communication and collaboration between the customer and the developer
- Elicitation—elicit requirements from all stakeholders
- Elaboration—create an analysis model that identifies data, function and behavioral requirements
- Negotiation—agree on a deliverable system that is realistic for developers and customers

Contd.,

- Specification—can be any one (or more) of the following:
  - A written document
  - A set of models
  - A formal mathematical
  - A collection of user scenarios (use-cases)
  - A prototype
- Validation—a review mechanism that looks for
  - errors in content or interpretation
  - areas where clarification may be required
  - missing information
  - inconsistencies (a major problem when large products or systems are engineered)
  - conflicting or unrealistic (unachievable) requirements.
- Requirements management

Establish the Ground Work

- **Identify Stakeholders**

A stakeholder as " anyone who benefits in a direct or indirect way from the system which is being developed.

- **Recognizing multiple viewpoints**
  - The Marketing group
  - Business Mangers
  - Software Engineers
  - Support Engineers
- **Working toward collaboration**
- **Asking the First Questions**

Eliciting Requirements

Requirements elicitation (also called requirements gathering) combines elements of problem solving, elaboration, negotiation, and specification. In order to encourage a collaborative , team-oriented approach to requirements gathering, stakeholders work together to identify the problem, propose elements of the solution, negotiate different approaches and specify a preliminary set of solution requirement.

- **Collaborative Requirements Gathering.**
- **Quality Function Deployment**
- **Usage Scenarios**
- **Elicitation work products**

Collaborative Requirements Gathering.

SAFEHOME

Conducting a Requirements Gathering Meeting

The scene: A meeting room. The first requirements gathering meeting is in progress.

The players: Jamie Lazar, software team member; Vinod Raman, software team member; Ed Robbins, software team member; Doug Miller, software engineering manager; three members of marketing; a product engineering representative; and a facilitator.

The conversation:

Facilitator (pointing at whiteboard): So that's the current list of objects and services for the home security function.

Marketing person: That about covers it from our point of view.

Vinod: Didn't someone mention that they wanted all *SafeHome* functionality to be accessible via the Internet? That would include the home security function, no?

Marketing person: Yes, that's right . . . we'll have to add that functionality and the appropriate objects.

Facilitator: Does that also add some constraints?

Jamie: It does, both technical and legal.

Production rep: Meaning?

Jamie: We better make sure an outsider can't hack into the system, disarm it, and rob the place or worse. Heavy liability on our part.

Doug: Very true.

Marketing: But we still need that . . . just be sure to stop an outsider from getting in.

Ed: That's easier said than done and . . .

Facilitator (interrupting): I don't want to debate this issue now. Let's note it as an action item and proceed.

(Doug, serving as the recorder for the meeting, makes an appropriate note.)

Facilitator: I have a feeling there's still more to consider here.

(The group spends the next 20 minutes refining and expanding the details of the home security function.)

Usage Scenarios

SAFEHOME

Developing a Preliminary User Scenario

The scene: A meeting room, continuing the first requirements gathering meeting.

The players: Jamie Lazar, software team member; Vinod Raman, software team member; Ed Robbins, software team member; Doug Miller, software engineering manager; three members of marketing; a product engineering representative; and a facilitator.

The conversation:

Facilitator: We've been talking about security for access to *SafeHome* functionality that will be accessible via the Internet. I'd like to try something. Let's develop a usage scenario for access to the home security function.

Jamie: How?

Facilitator: We can do it a couple of different ways, but for now, I'd like to keep things really informal. Tell us (he points at a marketing person) how you envision accessing the system.

Marketing person: Um . . . well, this is the kind of thing I'd do if I was away from home and I had to let someone into the house, say a housekeeper or repair guy, who didn't have the security code.

Facilitator (smiling): That's the reason you'd do it . . . tell me how you'd actually do this.

Marketing person: Um . . . the first thing I'd need is a PC. I'd log on to a website we'd maintain for all users of *SafeHome*. I'd provide my user id and . . .

Vinod (interrupting): The Web page would have to be secure, encrypted, to guarantee that we're safe and . . .

Facilitator (interrupting): That's good information, Vinod, but it's technical. Let's just focus on how the end user will use this capability. OK?

Vinod: No problem.

Marketing person: So as I was saying, I'd log on to a website and provide my user ID and two levels of passwords.

Questions:

1. Explain the requirements engineering process with help of a diagram? And also explain the spiral model of requirements?
2. Explain in detail on requirements Elicitation and Analysis process?
3. Elaborate on requirements Discovery and view points?
4. What are the goals of requirements engineering process?

Session – 8

Building Requirements Model

Elements of the Requirements Model

The specific elements of the requirements model are dictated by the analysis modeling method that is to be used. However, a set of generic elements is common to most requirements models.

- **Scenario-based elements**
- **Class-based elements**
- **Behavioral elements**
- **Flow-oriented elements**
- **Analysis Patterns**

Scenario-based elements

Use Case Diagram for Safe home

Activity Diagram for safe home

Class-based elements

Class Diagram for Sensor

Negotiating Requirements

Boehm [Boe98] defines a set of negotiation activities at the beginning of each software process iteration. Rather than a single customer communication activity, the following activities are defined:

1. **Identification of the system or subsystem's key stakeholders.**
2. **Determination of the stakeholders' "win conditions."**
3. **Negotiation of the stakeholders' win conditions to reconcile them into a set of win-win conditions for all concerned (including the software team).**

Validating Requirements

Verification
- designing the product right

Validation
- designing the right product

The formality gap
- validation will always rely to some extent on subjective means of proof

Management and contractual issues
- design in commercial and legal contexts

Validating Requirements

In the Validation process few questions should be asked and answered to ensure that the requirements model is an accurate reflection of stakeholder needs and that it provides a solid foundation for design.

Validating Requirements

1. Is each requirement consistent with the overall objectives for the system/product?
2. Have all requirements been specified at the proper level of abstraction? That is, do some requirements provide a level of technical detail that is inappropriate at this stage?
3. Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
4. Is each requirement bounded and unambiguous?
5. Does each requirement have attribution? That is, is a source (generally, a specific individual) noted for each requirement?

Cont..

6. Do any requirements conflict with other requirements?
7. Is each requirement achievable in the technical environment that will house the system or product?
8. Is each requirement testable, once implemented?
9. Does the requirements model properly reflect the information, function, and behavior of the system to be built?
10. Has the requirements model been “partitioned” in a way that exposes progressively more detailed information about the system?
11. Have requirements patterns been used to simplify the requirements model? Have all patterns been properly validated? Are all patterns consistent with customer requirements?

SRS Vs User Stories

User Stories

- Provide a small scale and easy to use presentation of information. Are generally formulated in the everyday language of the user and contain little detail, thus remaining open to interpretation. They should help the reader understand what the software should accomplish.
- Must be accompanied by acceptance testing procedures for clarification of behavior where stories appear ambiguous.

SRS

Software requirements specification establishes the basis for an agreement between customers and contractors or suppliers on how the software product should function.

Questions

1. Develop a complete use case for one of the following activities:
   a. Making a withdrawal at an ATM
   b. Using your charge card for a meal at a restaurant
   c. Buying a stock using an on-line brokerage account
   d. Searching for books (on a specific topic) using an on-line bookstore.
2. Develop a checklist for conducting a requirements gathering meeting.
3. Why do we say that the requirements model represents a snapshot of a system in time?
4. Why is it that many software developers don't pay enough attention to requirements engineering? Are there ever circumstances where you can skip it?
5. Discuss some of the problems that occur when requirements must be elicited from three or four different customers.
6. Define the metrics for specifying the NON-functional Requirements.
7. Discuss in detail the user Requirements?
8. Elaborate on software requirements document in detail?

Session – 9

Agile

Agile Development

What is “Agility”?

- Effective (rapid and adaptive) response to change
- Effective communication among all stakeholders
- Drawing the customer onto the team
- Organizing a team so that it is in control of the work performed

Yielding …

- Rapid, incremental delivery of software

Agile Development

Agile development methods apply time boxed iterative and evolutionary development, adaptive planning, promote evolutionary delivery, and include other values and practices that encourage agility.

An Agile Process

- Is driven by customer descriptions of what is required (scenarios)
- Recognizes that plans are short-lived
- Develops software iteratively with a heavy emphasis on construction activities
- Delivers multiple ‘software increments’
- Adapts as changes occur

Classification Methods

The Manifesto for Agile Software Development

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- ***Individuals and interactions* over processes and tools**
- ***Working software* over comprehensive documentation**
- ***Customer collaboration* over contract negotiation**
- ***Responding to change* over following a plan**

That is, while there is value in the items on the right, we value the items on the left more."

Agility and the Cost of Change

Agility Principles - I

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face–to–face conversation.

Agility Principles - II

7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential.
11. The best architectures, requirements, and designs emerge from self–organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile Project Management

Agile Project Management (APM) is an iterative approach to planning and guiding project processes.

Agile Project Management

Jim Highsmith, an Agile Alliance founder and creator of the Adaptive Software Development method, summarizes nine principles for the agile project manager [Highsmith02]:

1. Deliver something useful to the client; check what they value.
2. Cultivate committed stakeholders.
3. Employ a leadership-collaboration style.
4. Build competent, collaborative teams.
5. Enable team decision making.
6. Use short time boxed iterations to quickly deliver features.
7. Encourage adaptability.
8. Champion technical excellence.
9. Focus on delivery activities, not process-compliance activities.

Questions

1. Define Agility? Explain Agile process.
2. Explain Classification of methods in agile development.
3. List out Agile principles.
4. Illustrate the agile manifesto.
5. explain agile project management with help of Safe Home application

Session – 10

Agile

Topics

- Simple Practices and Project Tools
- Empirical vs. Defined & Prescriptive Process
- Principle-Based versus Rule-Based
- Agile Hype?
- Specific Agile Methods

Simple Practices and Project Tools

Simplicity—the art of maximizing the amount of work not done—is essential.

List Of The Best Agile Project Management Tools

Monday.com
Nifty
Wrike
SpiraTeam
Atlassian Jira
Active Collab
Agilo for Scrum
SpiraTeam
Pivotal Tracker
VSTS
Icescrum
Gravity
SprintGround
VersionOne
Taiga

Empirical vs. Defined & Prescriptive Process

In general, agile methods promote empirical rather than defined processes.

A defined process (also known as a **prescriptive process**) are suitable for predictable manufacturing domains.

Empirical processes are used for high-change and unstable domains.

Principle-Based vs. Rule-Based

Agile methods are more principle-based than rule-based.

Agile Methods

- Scrum
- XP
- Crystal Methods

Other Methods and Practices

- Adaptive Software Development
- Dynamic Solution Delivery Model
- Feature Driven Development
- Lean Development
- Pragmatic Programming

Agile Modeling

Questions

1. List out Various Agile project Management Tools and explain working functionality.
2. Differentiate between Empirical processes and defined process
3. Define principle based approach and rule based approach?
4. Agile Methods are Invent Iterative Development or not? Justify.
5. Write Short notes about Agile methods.
6. Compare and contrast Scrum and XP.

Session – 11

Extreme Programming

Extreme Programming

Extreme Programming (XP) is a well-known agile method; it emphasizes collaboration, quick and early software creation, and skillful development practices.

Extreme Programming

XP founded on four values:

communication, simplicity, feedback, and courage.

In addition to IID, it recommends **12 core practices**:

1. **Planning Game**
2. **small, frequent releases**
3. **system metaphors**
4. **simple design**
5. **testing**
6. **frequent refactoring**
7. **pair programming**
8. **team code ownership**
9. **continuous integration**
10. **sustainable pace**
11. **whole team together**
12. **coding standards**

Extreme Programming (XP) Model

- XP Planning
  - Begins with the creation of "user stories"
  - Agile team assesses each story and assigns a cost
  - Stories are grouped to for a deliverable increment
  - A commitment is made on delivery date
  - After the first increment "project velocity" is used to help define subsequent delivery dates for other increments

Extreme Programming (XP)

XP Design

- Follows the KISS(Keep it Simple, Stupid) principle
- Encourage the use of CRC cards (Class Responsibility Collaboration)
- For difficult design problems, suggests the creation of "spike solutions"—a design prototype(A Working Model)
- Encourages "refactoring"—an iterative refinement of the internal program design

XP Coding

- Recommends the construction of a unit test for a store *before* coding commences
- Encourages "pair programming"

XP Testing

- All unit tests are executed daily
- "Acceptance tests" are defined by the customer and executed to assess customer visible functionality

Extreme Programming (XP)

Extreme Programming (XP)

Method Overview

Work products Roles and Practices

Work Products (Non Software):

-Requirements – Story Cards

-Design- CRC Cards, Sketches

-Project Management- Task list, Visible graphs

Roles:

-Custumer

-Development

-Management

-Other

Practices:

-Requirements

-Design

-Implementation

-Test and Verification

-Project Management

-Configuration & Change management environment

Questions

1. List out Core practices in Extreme Programming.
2. Explain Extreme programming Life cycle.
3. Illustrate XP Work products Roles and Practices for safe home application.
4. Write extreme programming advantages and disadvantages?
5. How User requirements are expressed in Extreme Programming.

Session – 12

Extreme Programming

Extreme Programming

Adoption Strategies: XP recommends adoption like this

- Pick the worst project or problem.
- Apply XP until solved.
- Repeat.

If all the XP practices can't be swallowed at once, Beck recommends starting with:

- whole team together in a common project room
- test-first development
- acceptance tests written/owned by customers
- Planning Game
- pair programming

Extreme Programming

Fact versus Fantasy

Process is only a second-order effect. The unique people, their feelings and qualities, are more influential.

Extreme Programming

Strengths versus "Other"

Strengths:

- Practical, high-impact development techniques, many of which are easily and sustainably adopted by developers (e.g., continuous integration, test-driven development).
- Emphasizes customer participation and steering.
- Evolutionary and incremental requirements and development, and adaptive behavior.
- Programmers estimate the tasks they have chosen, and the schedule follows this, not vice versa (i.e., scheduling is rational).
- Emphasizes communication between all stakeholders.

- Emphasizes quality through many practices.
  Test-first development, continuous integration, and
  team code ownership are examples.
- Clarifies what is an acceptable system by requiring the customer to define the acceptance tests.
- Daily measurement, and developers are involved in measuring and defining what to measure.
- Every iteration, developers get practice (during the Planning Game) identifying tasks and estimating them, leading to improvement in these vital skills.
- Frequent, detailed reviews and inspections, as all significant work is done in pairs. Inspection is strongly correlated with reduced defect levels.

Extreme Programming

Strengths versus "Other"

Other:

- Requires the presence of onsite customers (or proxies).
- Relies on oral history for knowing the design and requirements details.
- The XP practices are interdependent and mutually supporting.
- No standard way to describe or document the software design as a learning aid.
- Some developers do not want to do pair programming.
- Many projects will need a set of documents other than code. XP does not define what these may be, and thus each project may create ones with similar intent, but varying names and content.
- Lack of architecture-oriented emphasis in the early iterations. Lack of architectural design methods. XP advocates claim simple design and refactoring lead to the same goal.

Questions

1. Write History of Extreme Programming.
2. List out Strengths of the Extreme Programming.
3. Identify activities involved in Extreme Programming.
4. Differentiate pair programming with normal programming.
5. List out Adoption Strategies in XP

Agile Development

Dr. S. Sri Harsha
Associate Professor
Empid:5887
Mail to :
sharsha@kluniversity.in

AGILE DEVELOPMENT

- Stake holders

Project manager

Software Developers

Customers

Writers

Consultants

What is it?

Agile Software Engineering

Philosophy

- Customer satisfaction
- Early incremental delivery of software
- Small, highly motivated project teams informal methods
- Minimal software engineering work products
- Overall development simplicity

A set of development guidelines

set of development guidelines:

- Delivery over analysis and design
- Active and continuous communication between developers & customers

Who does it?

- Software engineer and other project stakeholders(managers, customers, end users) working together on an agile team.

Why is it important?

- Agile software engineering represents a reasonable alternative approach to conventional software engineering for certain classes of software and software projects.
- To deliver successful systems quickly.

What is the work product?

- Software increment that is deliver to the customer on appropriate dates.

How do I ensure that I've done it right?

- If the agile team agrees that the process works and the team produces deliverable software increments that satisfy the customer.

- Revolutionary change
- Agile methods sometimes referred as light methods or lean methods.
- Market conditions change rapidly.
- End-user needs
- New competitive

What is agility?

Effective response to change.

Changes in the software being built

Changes to the team members

Changes because of new technology.

Agile Principles

- Satisfy the customer
- Welcome changing requirements, even late in the development.
- Deliver working software frequently from a couple of weeks to couple of months.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals give them the environment and support they need and trust them to get the job done.

- Face-to-Face conversation
- Working software is the primary measure of progress.
- Agile processes promote sustainable development, sponsors, developers and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity the art of maximizing the amount of work not done
- The best architectures, requirements & design emerge from self organizing teams

The Politics of Agile Development

- What is the best way to achieve it.
- How do you build software that meets customer needs today and exhibits the quality characteristics.

Human Factors

- Agile development focus on the talents and skills of individuals, molding the process to specific people & teams
- Competence
- Common focus
- Collaboration
- Decision-making ability
- Fuzzy problem-solving ability
- Mutual trust & respect
- Self-organization

Competence

- In agile development, talent, software-related skills and overall knowledge of the process that the team has chosen to apply.

- Skill and knowledge of process should be taught to all people who serve as agile team members

Common Focus

- All team members should be focused on one goal to deliver a working software increment to the customer within scheduled time.

Collaboration: assessing, analyzing, and using information that is communicated to the software team.

- Creating information that will help all stakeholders understand the work of the team.

Fuzzy problem-solving ability

The problem they are solving today may not be the problem that needs to be solved tomorrow.

Mutual trust and respect

“Jelled’’ team exhibits the trust and respect that are necessary to make them “so strongly knit that the whole is greater than the sum of the parts”

Self-organization

- The agile team organizes itself for the work to be done
- The agile team organizes the process to best accommodate its local environment
- The agile team organizes the work schedule to best achieve delivery of the software increment.

Extreme Programming

Dr. S. Sri Harsha
Associate Professor
Empid:5887
sharsha@kluniversity.in

Content

Extreme Programming

- XP Values
- The XP Process
- Industrial XP
- The XP Debate

Extreme Programming

- “XP is a light weight methodology for small to medium sized teams developing software in the face of vague or rapidly changing requirements”

Kent Beck

When applicabe

- Dynamically changing software requirements
- Risks caused by fixed time projects using new technology
- Small, co-located extended development team
- The technology you are using allows for automated unit and functional tests

XP Values

- Communication
- Simplicity
- Feedback
- Courage
- respect

Communication

- In order to achieve effective communication between software engineers and other stakeholders.
- XP emphasizes informal collaboration between customers and developers
- The establishment of effective metaphors for communicating important concepts, continuous feedback and the avoidance of voluminous documentation as a communication medium.
- Metaphor: the story that everyone- customers, programmers, and managers can tells about how the system works.

Simplicity

- To achieve simplicity, XP restricts developers to design only for immediate needs rather than future needs.
- To create a simple design that can be easily implemented in code.
- If the design must be improved, it can be refactored at a later time
- Refactoring-allows a software engineer to improve the internal structure of a design without changing its external functionality or behavior.
- Refactoring can be used to improve the efficiency, readability, or performance of a design or code that implements a design.

Feedback

- Feedback is derived from three sources:
- The implemented software itself
- The customer
- Other software team members
- Unit test
- By designing and implementing the effective testing strategy, software provides the agile team with feedback
- Increment is delivered to a customer, the users stories that are implemented by the increment are used as a basis for acceptance test.
- The degree to which the software implements the output, function and behavior of the use case is a form of feedback.
- Finally, as new requirements are derived as part of iterative planning, the team provides the customer with rapid feedback regarding cost and schedule time.

Courage

- Discipline
- There is often a significant pressure to design for future requirements.
- Most software teams succumb, arguing that “designing for tomorrow” will save time and effort in the long run.
- An agile xp team must have the discipline to design for today.

Respect

- The agile team inculcates respect among it members, between other stakeholders and team members and indirectly for the software itself.

Pair Programming

- Pair Programming means all production software is developed by two people sitting at the same machine. The idea behind this practice is that two brains and four eyes are better than one brain and two eyes. You effectively get a continuous code review and quicker response to nagging problems that may stop one person dead in their tracks.
- Teams that have used pair programming have found that it improves quality and does not actually take twice as long because they are able to work through problems quicker and they stay more focused on the task at hand, thereby creating less code to accomplish the same thing.

Stories

- Describe what the product should do in terms meaningful to customers and users. These stories are intended to be short descriptions of things users want to be able to do with the product that can be used for planning and serve as reminders for more detailed conversations when the team gets around to realizing that particular story.

XP Process

- Planning
- Design
- Coding
- Testing

Planning

- Listening-creation of user stories
- Requirement gathering activity
- User stories written by customer and is placed on index card
- Xp team orders the stories that will be developed in 3 ways
- All user stories will be implemented immediately within a few weeks
- The stories with highest value will be moved up in the schedule and implemented first.
- The riskest stories will be moved up in the schedule and implemented first.

- Project velocity-number of customer stories implemented during the first release.
- Project velocity can be done in 3 ways
- Estimate delivery dates and schedule for subsequent releases
- Overcommitment has been made for all stories across the entire development project

Design

- XP team follows the KIS principle(Keep it Simple)
- A simple design is always preferred over a complex design.
- Design provides implementation guidance for a story as it is written.
- CRC- Class Responsibility Collaboration
- XP Team encourages the use of CRC cards as an effective mechanism for thinking about software in object-oriented context .
- CRC cards identify object oriented classes that are relevant to current software increment.

Coding

- After user stories are developed and preliminary design work is done, the team does not move to code but rather than develops a series of unit tests that will exercise each of the stories that is to be included in the current release.
- Unit Testing: test individual components of a software.

Testing

- Unit testing
- Integration testing
- Acceptance testing
- Acceptance testing also known as customer tests, are specified by customer and focus on overall system features and functionality that are visible and reviewable by the customer.
- Acceptance tests are derived from user stories that have been implemented as part of a software release

Industrial XP

- Readiness assessment
- Project community
- Project chartering
- Test driven management
- Retrospectives
- Continuous learning

Readiness assessment

Prior to the initiation of an IXP project, the organization should conduct a readiness assessment.

1) An appropriate development environment exists to support IXP.
2) The team will be populated by the proper set of stakeholders
3) The organization has a distinct quality program and supports continuous improvement.
4) The organization culture will support the new values of an agile team
5) The broader project community.

Project community

Classical xp

- Right people used in agile team to ensure success.
- The team must be well-trained, adaptable and skilled and have proper temperament to contribute to a self-organizing team.

IXP

- A community may have technologists and customers who are central to the success of the project.
- Other stakeholders(legal staff, quality auditors, manufacturing or sales types) may play a important roles on the project.

Project chartering

Examine the context of the project how it complements, extends or replaces existing systems.

The project charter typically documents:

- Reasons for the project
- Objectives and constraints of the project
- Who the main stakeholders are
- Risks identified
- Benefits of the project
- General overview of the budget

Test-driven management

Measurable criteria for assessing the state of the project and the progress that has been made to date.

Series of measurable “destinations” and then defines mechanisms for determining whether these destinations reached or not.

Retrospectives.

IXP team conducts a Special technical review after software increment is delivered.

Issues

Events

Lessons-learned across the software increment

The goal is to improve the IXP process.

Continuous Learning:

Learning is a continuous process improvement

Members of xp team are encouraged to learn new methods and techniques to lead a high quality software product.

XP Debate

The issues that continue to trouble some critics of XP are:

- Requirements volatility
- Conflicting customer needs
- Requirements are informally
- Lack of formal design

Other Agile Process models

- Adaptive Software Development
- Scrum
- Dynamic System Development Method
- Crystal
- Feature Driven Development
- Lean Software Development

Scrum

- Agile software development method
- Frame work activities
- Requirements
- Analysis
- Design
- Evolution
- Delivery

- The overall flow of the scrum process
- Backlogs
- Sprints
- Scrum meeting
- Three key questions asked and answered by all the team members
- What did you do since the last team meeting
- What obstacles are you encountering
- What do you plan to accomplish by the next meeting
- Scrum master
- demos

Session – 13
Scrum

K L University

u/s 3 of UGC Act. 1956

Koneru Lakshmaiah Education Foundation

Scrum Introduction

- Scrum is a process framework used to manage product development and other knowledge work.
- Scrum is empirical in that it provides a means for teams to establish a hypothesis of how they think something works, try it out, reflect on the experience, and make the appropriate adjustments. That is, when the framework is used properly.
- Scrum is structured in a way that allows teams to incorporate practices from other frameworks where they make sense for the team's context.
- Scrum appears simple, yet has practices that deeply influence the work experience and that capture key adaptive and agile qualities.

What is Scrum?

Scrum:

- Is an agile, lightweight process
- Can manage and control software and product development
- Uses iterative, incremental practices
- Has a simple implementation
- Increases productivity
- Reduces time
- Embraces the opposite of the waterfall approach…

Scrum Principles

Scrum at a Glance

Source: Adapted from *Agile Software Development with Scrum* by Ken Schwaber and Mike Beedle.

Scrum Framework

Roles

- Product owner
- Scrum Master
- ProjectTeam

Ceremonies

- Sprint planning
- Daily scrum meeting
- Sprint review
- Sprint retrospective

Artifacts/work products

- Product backlog
- Sprint backlog
- Burndown charts

Scrum Roles

- Product Owner
  - Possibly a Product Manager or Project Sponsor
  - Decides features, release date, prioritization, $$$

- Scrum Master
  - Typically a Project Manager or Team Leader
  - Responsible for enacting Scrum values and practices
  - Remove impediments / politics, keeps everyone productive

- Project Team
  - 5-10 members; Teams are self-organizing
  - Cross-functional: QA, Programmers, UI Designers, etc.
  - Membership should change only between sprints

Scrum Ceremonies

Sprint Planning Mtg.

Daily Scrum Meeting

- Parameters
  - Daily, 15 minutes, Stand-up
  - Anyone late pays a $1 fee

- Not for problem solving
  - Whole world is invited
  - Only team members, Scrum Master, Product owner, can talk
  - Helps avoid other unnecessary meetings

- Three questions answered by each team member:
  1. What did you do yesterday?
  2. What will you do today?
  3. What obstacles are in your way?

Sprint Review

- Sprint Review is held at the end of the Sprint to inspect the Increment and adapt the Product Backlog if needed.

- Attendees: Scrum Team and key stakeholders;
- Product Owner: explains what Product Backlog items have been "Done" and what has not been "Done"; he projects likely target and delivery dates based on progress to date (if needed);
- Development Team: what went well, what problems it ran into, and how those problems were solved;
- The Development Team demonstrates the work that it has "Done" and answers questions about the Increment;
- The Product Owner or Team or entire group collaborates on what to do next, so that the Sprint Review provides valuable input to subsequent Sprint Planning

Sprint Retrospective

Scrum's Artifacts/Work products

- Scrum has remarkably few artifacts
  - Product Backlog
  - Sprint Backlog
  - Burndown Charts

- Can be managed using just an Excel spreadsheet
  - More advanced / complicated tools exist:
    - Expensive
    - Web-based – no good for Scrum Master/project manager who travels
    - Still under development

Product Backlog

- The requirements
- A list of all desired work on project
- Ideally expressed as a list of user stories along with "story points", such that each item has value to users or customers of the product
- Prioritized by the product owner
- Reprioritized at start of each sprint

User Stories

- Instead of Use Cases, Agile project owners do "user stories"
  - **Who** (user role) – Is this a customer, employee, admin, etc.?
  - **What** (goal) – What functionality must be achieved/developed ?
  - **Why** (reason) – Why does user want to accomplish this goal?

    As a [user role], I want to [goal], so I can [reason].
- Example:
  - "As a user, I want to log in, so I can access subscriber content."

- **story points**:is a metric used to estimate the difficulty of implementing a given user story. A number that tells the team about the difficulty level of the **story**.
- Rating of effort needed to implement this story
  - common scales: 1-10, shirt sizes (XS, S, M, L, XL), etc.

Sprint Backlog

- Individuals sign up for work of their own choosing
  - Work is never assigned
- Estimated work remaining is updated daily
- Any team member can add, delete change sprint backlog
- Work for the sprint emerges
- If work is unclear, define a sprint backlog item with a larger amount of time and break it down later
- Update work remaining as more becomes known

Burndown Chart

- It is a visual measurement tool that shows the completed work per day against the projected rate of completion for the current project release. Its purpose is to enable that the project is on the track to deliver the expected solution within the desired schedule.

Questions

1. Define SCRUM.
2. Explain Method Overview of scrum.
3. Sketch and explain Lifecycle of scrum.
4. Explain Workproducts, Roles, and Practices.

End of Session 13

Session – 14

Scrum Cont..

K L University

u/s 3 of UGC Act. 1956

Koneru Lakshmaiah Education Foundation

The Scrum Meeting: Details

- The Scrum Meeting—or scrum—is the heartbeat of Scrum and the project. Each workday at the same time and place, hold a meeting with the team members *standing* in a circle, at which time the same special questions are answered by each member:

1. What have you done since the last Scrum?

2. What will you do between now and the next Scrum?

3. What is getting in the way (blocks) of meeting the iteration goals?

4. Any tasks to add to the Sprint Backlog

5. The last question provides an efficient forum for a continuously improving and learning group

Some key practices

- *self-directed and self-organizing team*
- *no external addition of work to an iteration, once chosen*
- *daily stand-up meeting with special questions*
- *usually 30-calendar day iterations*
- *demo to external stakeholders at end of each iteration*
- *each iteration, client-driven adaptive planning*

Other Practices and Values

Other Practices:

- Workers daily update the Sprint Backlog
- No PERT charts allowed
- Scrum Master reinforces vision
- Replace ineffective Scrum Master

Scrum Values:

- Commitment: Team members personally commit to achieving team goals
- Courage: Team members do the right thing and work on tough problems.
- Focus: Concentrate on the work identified for the sprint and the goals of the team.
- Openness: Team members and stakeholders are open about all the work and the challenges the team encounters.
- Respect: Team members respect each other to be capable and independent.

Common Mistakes and Misunderstandings

- Not a self-directed team; managers or Scrum Master direct or organize the team
- No daily update of the Sprint Backlog by members or daily tracker
- New work added to iteration or individual
- Product Owner isn't involved or doesn't decide
- No Sprint Review
- Many masters
- Documentation is bad
- Design or diagramming is bad
- Full team (including customers and management) not briefed in Scrum and its values
- Scrum Meeting too long or unfocused
- Predictive planning; PERT chart planning

Adoption Strategies

12 Must do's for successful adoption of Agile Scrum methodology:

- Assess Project for readiness & suitability
- Identify a 'Go-Get-It' team
- Pilot when maximum teams are ready to fly
- Convince the party poopers
- Be Patient while transformation
- Set a high bar and low expectations
- Handhold to smoothen friction
- Be open to Experts' help
- Make Good Information More Accessible
- Find Your Evangelists
- Measure the Results Early and Often
- Be sure what's Scrum and what's Not

The Sprint Review

- Team presents what it accomplished during the sprint
- Typically takes the form of a demo of new features or underlying architecture
- Informal
  - 2-hour prep time rule
  - No slides
- Whole team participates
- Invite the world

Sprint Review Cont:-

Scalability

- Typical individual team is 7 ± 2 people
  - Scalability comes from teams of teams
- Factors in scaling
  - Type of application
  - Team size
  - Team dispersion
  - Project duration
- Scrum has been used on multiple 500+ person projects

Scaling: Scrum of Scrums

Scrum of Scrums

- Scale **Scrum** up to large groups
- Consisting of dividing the groups into Agile teams of 5-10
- Each daily **scrum** within a sub-team ends by designating one member as “ambassador”
- “Ambassador” participate in a daily meeting with ambassadors from other teams

Purpose

- Scale the daily stand-up meeting when multiple teams are involved.
- Its **purpose** is to support **agile** teams in collaborating and coordinating their work with other teams.

Scrum vs. Other Models

Process Comparison

	Waterfall	Spiral	Iterative	SCRUM
Defined processes	Required	Required	Required	Planning & Closure only
Final product	Determined during planning	Determined during planning	Set during project	Set during project
Project cost	Determined during planning	Partially variable	Set during project	Set during project
Completion date	Determined during planning	Partially variable	Set during project	Set during project
Responsiveness to environment	Planning only	Planning primarily	At end of each iteration	**Throughout**
Team flexibility, creativity	Limited - cookbook approach	Limited - cookbook approach	Limited - cookbook approach	**Unlimited during iterations**
Knowledge transfer	Training prior to project	Training prior to project	Training prior to project	**Teamwork during project**
Probability of success	Low	Medium Low	Medium	**High**

Fact versus Fantasy

- First, a standard disclaimer: Process is only a second-order effect. The unique people, their feelings and qualities, are more influential.

- Scrum practitioners do not report significant variation from the ideals of Scrum compared to its concrete use, presumably due to the relatively small and unambiguous set of practices. The most commonly reported reality checks are the encroachment of non-iteration work on to team members, and attempts by management to direct or organize the team, or solve—unasked—its problems. Scrum iterations have also failed when the Scrum Master does not regularly reinforce the project vision and Sprint goals, and the team drifts.

Strengths versus "Other"

- Simple practices and management workproducts.
- Individual and team problem solving and self-management.
- Evolutionary and incremental requirements and development, and adaptive behavior.
- Customer participation and steering.
- Focus.
- Openness and visibility.
- Easily combined with other methods.
- Team communication, learning, and value-building.
- Team building via the daily Scrum, even if not in common project room.

Other: Could be viewed as a weakness, strength, or deliberate desirable exclusion, depending on point of view.

History of Scrum

Questions

1. Advantages of Scrum Meeting.
2. List out the values of scrum.
3. List out the Adoption Strategies.
2. Explain Process Mixtures of scrum.
3. List out the strengths of Scrum.
4. List out common mistakes and misunderstandings of scrum.

Session – 15

KANBAN

K L University

u/s 3 of UGC Act. 1956

Koneru Lakshmaiah Education Foundation

Topics to be discussed

- Where did Kanban originate?
- What is the Kanban Method?
- Kanban Foundational Principles

INTRODUCTION

KANBAN

- Kanban is a visual system for managing work.
- It Visualizes both the process (the workflow) and the actual work passing through that process.
- Kanban is a workflow management method designed to help us to visualize your work, maximize efficiency.
- The goal of Kanban is to identify potential bottlenecks in your process and fix them, so work can flow through it cost-effectively at an optimal speed or throughput.

From Japanese, kanban is literally translated as billboard or signboard. Originating from manufacturing industry.

.

Where did Kanban originate? – A Brief History on Kanban

- Kanban originated from the Toyota Production System (TPS). In the late 1940s, Toyota introduced “just in time” manufacturing to their production.
- The approach represents a pull system.
- This means that production is based on customer demand, rather than the standard push practice to produce amounts of goods and pushing them to the market.
- Its core purpose is minimizing waste activities without sacrificing productivity.
- The main goal is to create more value for the customer without generating more costs.

What is the Kanban Method ?

Kanban is not a software development lifecycle methodology or an approach to project management.

It requires that some process is already in place so that Kanban can be applied to incrementally change the underlying process.

The Kanban Method is a process to gradually improve whatever you do- almost any business function can benefit from applying the principles of the Kanban Methodology.

Kanban Principles & Practices

The four foundational principles

- Start with what you are doing now:
- Agree to pursue incremental, evolutionary change
- Initially, respect current roles, responsibilities and job-titles
- Encourage acts of leadership at all levels

Questions

1. Define Kanban.
2. Where did Kanban originate?
3. What is the Kanban Method?
4. Explain Kanban Foundational Principles

Session – 16

KANBAN

K L University

u/s 3 of UGC Act. 1956

Koneru Lakshmaiah Education Foundation

Topics to be discussed

- *Where did Kanban originate?*
- *What is the Kanban Method?*
- *Kanban Foundational Principles*
- *6 Core Practices of the Kanban*
- *Positive side of Kanban*
- *Main components of Kanban Board*
- *WIP Limits in Kanban*
- *Prioritizing the Kanban Backlog*

6 Core Practices of the Kanban Method

- Visualize the flow of work
- Limit WIP (Work in Progress)
- Manage Flow
- Make Process Policies Explicit
- Implement Feedback Loops
- Improve Collaboratively, Evolve Experimentally (using the scientific method)

Visualize the flow of work:

Limit work in progress

The Positive Side of Kanban

- Everyone is on the same page
- Kanban reveals bottlenecks in your workflow
- Kanban brings flexibility
- Your team gets more responsive
- You focus on finishing work to boost collaboration and productivity

Main Components of the Kanban board

- **Kanban Cards** – This is the visual representation of tasks. Each card contains information about the task and its status such as deadline, assignee, description, etc.
- **Kanban Columns** – Each column on the board represents a different stage of your workflow. The cards go through the workflow until their full completion.
- **Work-in-Progress Limits** – They restrict the maximum amount of tasks in the different stages of the workflow. Limiting WIP allows you to finish work items faster, by helping your team to focus only on current tasks.
- **Kanban Swimlanes** – These are horizontal lanes you can use to separate different types of activities, teams, classes of service,

What Is a Kanban Card?

A Kanban card contains valuable information about the task and its status such as a summary of the assignment, responsible person, deadline, etc.

An example of a physical Kanban card.

Kanban WIP Limits

- The acronym WIP stands for Work In Progress. WIP is the number of task items that a team is currently working on. It frames the capacity of your team’s workflow at any moment. Limiting work in progress is one of the core properties of Kanban. It allows you to manage your process in a way that creates smooth workflow and prevents overloads.

Prioritizing the Kanban Backlog

- The backlog is the space where you place work items or ideas that will be done in the near or distant future. However, there is no guarantee that all tasks in the Kanban Backlog will be delivered. The items in this column are more like an option the team has for the future work rather than a commitment point.

Prioritizing Tasks With Color Indicators

Questions

1. Define Kanban.
2. Where did Kanban originate?
3. What is the Kanban Method?
4. Explain Kanban Foundational Principles
5. List out the 6 Core Practices of the Kanban
6. How does Kanban work? – The Concept
7. What are WIP Limits in Kanban.

Session – 17

SAFe Methodology

K L University

u/s 3 of UGC Act. 1956

Koneru Lakshmaiah Education Foundation

INTRODUCTION

Scaled Agile Framework (SAFe)

The SAFe framework was introduced in 2011. It was originally called the “Agile Enterprise Big Picture”

The Scaled Agile Framework, or SAFe, methodology is an agile framework for development teams built on three pillars: Team, Program, and Portfolio.

What we Discuss

- What is Scaled Agile Framework (SAFe)
- Why to use Agile Framework
- When to Use Scaled Agile Framework
- Foundations of Scaled Agile Framework

Why to use Agile Framework

Agile Process Works

When to Use Scaled Agile Framework

Foundations of Scaled Agile Framework

SAFe Lean-Agile Principles

These basic principles and values for SAFe must be understood, exhibited and continued in order to get the desired results.

- *Take an economic view*
- *Apply systems thinking*
- *Build incrementally with fast, integrated learning cycles*
- *Base milestones on an objective evaluation of working systems*
- *Visualize and limit WIP, reduce batch sizes and manage queue lengths*
- *Decentralize decision-making*

SAFe Agile Core Values

The SAFe agile is based on these four values.

Alignment:

Built-in Quality

Transparency:

Program Execution:

Lean Agile Leaders

The Lean-Agile Leaders are lifelong learners and teachers.

It helps teams to build better systems through understanding and exhibiting the Lean-Agile SAFe Principles.

Lean Agile Mind-Set

Lean-Agile mindset is represented in two things:

1. The SAFe House of Lean

2. Agile Manifesto

Session – 18

SAFe Methodology

K L University

u/s 3 of UGC Act. 1956

Koneru Lakshmaiah Education Foundation

How different than other Agile practices

Let's see how Scaled Agile framework is different from other agile practices,

- It's publicly available and free to use.
- Available in a highly approachable and usable form.
- it constantly/regularly modifies/maintains most commonly used agile practices.
- Offers useful extensions to common agile practices.
- Grounds agile practices to an enterprise context.
- Offers complete picture of software development.
- Visibility or transparency is more on all the levels.
- Continues or regular feedback on quality and improvement.

Different Levels in SAFE

There are two different types of SAFe implementation:

1. SAFe 4.0 implementation .
2. SAFe 3.0 implementation.

Team Level

Roles/Teams	Events	Artifacts
* Agile Team	* Sprint Planning	* Team Backlog
* Product Owner	* Backlog Grooming	* Non-Functional Requirements
* Scrum Master	* Daily Stand-Up	* Team PI Objectives
	* Execution	* Iterations
	* Sprint Demo	* Stories(Working Software)
	* Sprint Retrospective	* Sprint Goals
	* IP Sprints	* Built-In Quality
		* Spikes
		* Team Kanban

Program Level

Roles/Teams	Events	Artifacts
* DevOps	* PI(Program Increment) Planning	* Vision
* System Team	* System Demos	* Roadmap
* Release Management	* Inspect and Adopt Workshop	* Metrics
* Product Management	* Architectural Runway	* Milestones
* UEX Architect	* Release Any Time	* Releases
* Release Train Engineer(RTE)	* Agile Release Train	* Program Epics
* System Architect/Engineer	* Release	* Program Kanban
* Business Owners		* Program Backlog
* Lean-Agile Leaders		* Non-Functional Requirements
* Communities of Practice		* Weighted Shortest Job First (WSJF)
* Shared Services		* Program PI Objectives
* Customer		* Feature
		* Enabler
		* Solution
		* Value Stream Coordination

Portfolio Level

Roles/Teams	Events	Artifacts
* Enterprise Architect	* Strategic Investment Planning	* Strategic Themes
* Program Portfolio Mgmt	* Kanban Portfolio(Epic) Planning	* Enterprise
* Epic Owners		* Portfolio Backlog
		* Portfolio Kanban
		* Non-Functional Requirements
		* Epic and Enabler
		* Value Stream
		* Budgets(CapEx and OpEx)

Value Stream Level

Roles/Teams	Events	Artifacts
* DevOps	* Pre and Post PI(Program Increment) Planning	* Vision
* System Team	* Solution Demos	* Roadmap
* Release Management	* Inspect and Adopt Workshop	* Metrics
* Solution Management	* Agile Release Train	* Milestones
* UEX Architect		* Releases
* Value Stream Engineer(RTE)		*Value Stream Epics
* Solution Architect/Engineer		* Value Stream Kanban
* Shared Services		* Value Stream Backlog
* Customer		* Non-Functional Requirements
* Supplier		* Weighted Shortest Job First (WSJF)
		* Value Stream PI Objectives
		* Capability
		* Enabler
		* Solution Context
		* Value Stream Coordination
		* Economic Framework
		* Solution Intent
		* MBSE
		* Set Based

Questions

1. How different than other Agile practices
2. List out the principles of Agile Manifesto
3. Explain Different Levels in SAFE

19CS2211 - Software Engineering

Software Testing Strategies

Session – 19

K L University

u/s 3 of UGC Act. 1956

Koneru Lakshmaiah Education Foundation

INTRODUCTION

- Software Testing Strategies
  - describes the steps to be conducted
  - effort, time, and resources will be required
  - test planning, test case design, test execution, and resultant data collection and evaluation
  - flexible enough to promote a customized testing approach.
  - reasonable planning and management tracking as the project progresses
- Testing - process - intent of finding errors prior to delivery to the end user.

What Testing Shows

A Strategic Approach To Software Testing

- To perform effective testing - technical reviews
- Testing begins at the component level
- Different testing techniques
- Testing is conducted – by independent group
- Testing and debugging are different activities

Verification and Validation

- Verification
  - to ensure that software correctly implements a specific function.

- Validation
  - to ensure task is traceable to customer requirements.

Verification: "Are we building the product right?"

Validation: "Are we building the right product?"

Organizing for Software Testing

- Developer:
  - testing the individual units (components)
  - ensuring that function behavior for which it was designed.
- Independent Test Group (ITG):
  - remove the inherent problems
  - removes the conflict of interest
  - ITG personnel are paid to find errors

Software Testing Strategy—The Big Picture

Testing Strategy

- Begin with "testing-in-the-small" and move toward "testing-in-the-large"

- For conventional software
  - The module (component) is first initial focus
  - Integration of modules follows

- For Object Oriented software
  - our focus when "testing in the small" changes from an individual module (the conventional view) to an OO class that encompasses attributes and operations and implies communication and collaboration

Strategic Issues

- Specify product requirements in a quantifiable manner long before testing commences.
- State testing objectives explicitly.
- Understand the users of the software and develop a profile for each user category.
- Develop a testing plan that emphasizes “rapid cycle testing.”
- Build “robust” software that is designed to test itself.
- Use effective technical reviews as a filter prior to testing.
- Conduct technical reviews to assess the test strategy and test cases themselves.
- Develop a continuous improvement approach for the testing process.

Test Strategies for Conventional Softwar

- A testing strategy incremental view of testing
  - Unit Testing
  - Integration Testing
- Unit Testing:
  - Verification on the smallest unit of software

Unit Testing

- Verification on the smallest unit of software
- Unit-test considerations:
  - ensure that information properly flows into and out of the program unit under test.
  - All independent paths through the control structure are exercised to ensure that all statements in a module have been executed at least once.
  - Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing.
  - All error-handling paths are tested.

Unit-test Procedures

- Unit testing is normally considered as an adjunct (i.e., extra) to the coding step.
- The design of unit tests can occur before coding begins or after source code has been generated.
- A review of design information provides guidance for uncovering.

Integration Testing

- It is a systematic technique for constructing the software architecture
- To uncover errors associated with interfacing.
- The objective is to take unit-tested components and build a program structure that has been dictated by design.
- **Top-down integration:**
  - Top-down integration testing is an incremental approach
  - Modules are integrated by moving downward through the control hierarchy, beginning with the main control module(main program).
  - Modules subordinate (and ultimately subordinate) to the main control module are incorporated into the structure in either a depth-first or breadth-first manner

– Depth-first integration integrates all components on a major control path of the program structure.

The integration process is performed in a series of five steps:

1. The main control module is used as a test driver and stubs are substituted for all components directly subordinate to the main control module.
2. Depending on the integration approach selected (i.e., depth or breadth first), subordinate stubs are replaced one at a time with actual components.
3. Tests are conducted as each component is integrated.
4. On completion of each set of tests, another stub is replaced with the real component.
5. Regression testing (discussed later in this section) may be conducted to ensure that new errors have not been introduced.

The process continues from step 2 until the entire program structure is built.

- Bottom-up integration:
  - begins construction and testing with atomic modules (i.e., components at the lowest levels in the program structure).

    1. Low-level components are combined into clusters (sometimes called builds) that perform a specific software subfunction.
    2. A driver (a control program for testing) is written to coordinate test case input and output.
    3. The cluster is tested.
    4. Drivers are removed and clusters are combined moving upward in the program structure.

M_c
M_a
M_b
D_1
D_2
D_3
Cluster 1
Cluster 2
Cluster 3

Regression Testing

- Each time a new module is added as part of integration testing, the software changes.
- New data flow paths are established, new I/O may occur, and new control logic is invoked.
- These changes may cause problems with functions that previously worked flawlessly.
- In the context of an integration test strategy, regression testing is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects.

Smoke Testing

- Smoke testing is an integration testing approach that is commonly used when product software is developed.
- Designed for time-critical projects, allowing the software team to assess the project on a frequent basis.

- The smoke-testing approach encompasses the following activities:
  - Software components -> code -> data files, libraries, reusable modules, and engineered components
  - Series of tests is designed - to expose errors.
  - The build (product) is integrated with other builds, and the entire product (in its current form) is smoke tested daily.
  - The integration approach may be top down or bottom up.

- Benefits of Smoke Test:
  - Integration risk is minimized.
  - The quality of the end product is improved.
  - Error diagnosis and correction are simplified.
  - Progress is easier to assess.

Revision Questions

1. Define Software Testing.
2. Explain A Strategic Approach To Software Testing in detail
3. Sketch how testing strategy is represented in spiral model and explain.
4. Explain in detail about Testing Strategy.
5. List out the Strategic Issues of Software testing.
6. Explain Smoke Test in detail.

Thank You

19CS2211 - Software Engineering

Software Testing

Session – 20

K L University

u/s 3 of UGC Act. 1956

Koneru Lakshmaiah Education Foundation

Black-box Testing

- Also called behavioral testing
- Focuses - functional requirements of software.
- Enable sets of input conditions - fully exercise all functional requirements for a program.
- Black-box testing is not an alternative to white -box techniques.
- To uncover a different class of errors.

- Tests are designed to answer the following questions:
  - How is functional validity tested?
  - How are system behavior and performance tested?
  - What classes of input will make good test cases?
  - Is the system particularly sensitive to certain input values?
  - How are the boundaries of a data class isolated?
  - What data rates and data volume can the system tolerate?
  - What effect will specific combinations of data have on system operation?

- By applying black-box techniques, you derive a set of test cases that satisfy the following criteria:
  - test cases that reduce, by a count that is greater than one, the number of additional test cases that must be designed to achieve reasonable testing, and
  - test cases that tell you something about the presence or absence of classes of errors, rather than an error associated only with the specific test at hand.

Graph-Based Testing Methods

- Understand the objects in software
- Connect these objects using relationships.
- Define a series of tests - all objects have the expected relationship to one another.
- *A number of behavioral testing methods that can make use of graphs:*
  - Transaction flow modeling.
  - Finite state modeling.
  - Data flow modeling.
  - Timing modeling.

(a) Graph notation
(b) simple example

Equivalence Partitioning

- It is a black-box testing method.
- Divides the input domain into classes of data
- from these data - test cases can be derived.
- Test-case design - based on an evaluation of equivalence classes for an input condition.

Equivalence classes may be defined according to the following guidelines:

1. If an input condition ***specifies a range***, one valid and two invalid equivalence classes are defined.
2. If an input condition ***requires a specific value***, one valid and two invalid equivalence classes are defined.
3. If an input condition ***specifies a member of a set***, one valid and one invalid equivalence class are defined.
4. If an input condition ***is Boolean***, one valid and one invalid class are defined.

Boundary Value Analysis (BVA)

- More errors occurs at the boundaries in input
- Lesser errors in “center” in input
- For this reason that boundary value analysis (BVA) has been developed as a testing technique.
- BVA leads to a selection of test cases that exercise bounding values.

- **Guidelines for BVA are similar in many respects to those provided for equivalence partitioning:**
  - If an input condition **specifies a range bounded by values a and b,** test cases should be designed with values a and b and just above and just below a and b.
  - If an input condition **specifies a number of values**, test cases should be developed that exercise the minimum and maximum numbers. Values just above and below minimum and maximum are also tested.

- **Apply guidelines 1 and 2 to output conditions**. Test cases should be designed to create an output report that produces the maximum (and minimum) allowable number of table entries.
- If internal program data structures have prescribed boundaries (e.g., a table has a defined limit of 100 entries), be certain to design a test case to exercise the data structure at its boundary.

Orthogonal Array Testing

- Applied to problems in which the input domain is relatively small but too large to accommodate exhaustive testing.
- Useful in finding region faults - faulty logic within a software component.

Black-box Testing

- To find errors in the following categories:
  - incorrect or missing functions
  - interface errors
  - errors in data structures or external database access
  - behavior or performance errors
  - initialization and termination errors

White-box Testing

- Called glass-box testing
- Used in control structure
  - guarantee that all **independent paths** within a module have been exercised at least once
  - exercise all **logical decisions** on their true and false sides
  - execute **all loops at their boundaries** and within their operational bounds
  - exercise **internal data structures** to ensure their validity

Validation Testing

- It begins at the culmination of integration testing
- when individual components have been exercised, the software is completely assembled as a package, and interfacing errors have been uncovered and corrected.
- At the validation or system level, the distinction between conventional software, object-oriented software, and WebApps disappears.
- Testing focuses on user-visible actions and user-recognizable output from the system.

- **Validation Criteria:**
- After each validation test case has been conducted, one of two possible conditions exists:
  - The function or performance characteristic conforms to specification and is accepted **or**
  - a deviation from specification is uncovered and a deficiency list is created.

Note:

- Deviations or errors discovered at this stage in a project can rarely be corrected prior to scheduled delivery.

Configuration Review:

To ensure that all elements of the software configuration have been properly developed, are cataloged, and have the necessary detail to bolster the support activities.

Alpha and Beta Testing

- The alpha test:
  - Conducted at the developer's site
  - Developer should present
  - Used in a natural setting with the developer "looking over the shoulder" of the users and recording errors and usage problems.
- The beta test:
  - Conducted at one or more end-user sites.
  - Developer generally is not present.
  - Therefore, the beta test is a "live" application of the software in an environment that cannot be controlled by the developer.

- A variation on beta testing, called customer acceptance testing:
  - Performed when software is delivered to a customer under contract.
  - The customer performs a series of specific tests in an attempt to uncover errors before accepting the software from the developer.

System Testing

- series of different tests whose primary purpose is to fully exercise the computer-based system.
- Purpose - to verify that system elements have been properly integrated and perform allocated functions.

Recovery Testing:

- It is a system test
- It forces the software to fail in a variety of ways and verifies that recovery is properly performed.
- If recovery is automatic (performed by the system itself), re-initialization, check pointing mechanisms, data recovery, and restart are evaluated for correctness.

Security Testing:

- Protect the system from improper penetration.

Stress Testing:

- Executes a system – with resources in abnormal quantity, frequency, or volume.
- A variation of stress testing is a technique called ***sensitivity testing***.
- Sensitivity testing - to uncover data combinations within valid input classes that may cause instability or improper processing.

Performance Testing:

- To test the run-time performance of software within the context of an integrated system.
- Performance testing occurs throughout all steps in the testing process.

  Performance tests are often coupled with stress testing
- It require both hardware and software instrumentation.

Deployment Testing:

- Called configuration testing
- Exercises the software in each environment in which it is to operate
- Examines all installation procedures and specialized installation software (e.g., “installers”) that will be used by customers
- All documentation that will be used to introduce the software to end users

Revision Questions

1. Define System Testing.
2. Outline the types of System testing.
3. Explain behavioral testing that make use of graphs.
4. Distinguish White Box and Black Box Testing.
5. Distinguish Validation Testing and Black Box Testing.
6. Distinguish Performance Testing and Deployment Testing.

Thank You

19CS2211 - Software Engineering

Software Testing – Black-box Testing

Session – 21

K L University

u/s 3 of UGC Act. 1956

Koneru Lakshmaiah Education Foundation

Black-box Testing

- It is called as behavioral testing
- Focuses - functional requirements of software.
- Enable sets of input conditions - fully exercise all functional requirements for a program.
- Black-box testing is not an alternative to white -box techniques.
- To uncover a different class of errors.

- Tests are designed to answer the following questions:
  - How is functional validity tested?
  - How are system behavior and performance tested?
  - What classes of input will make good test cases?
  - Is the system particularly sensitive to certain input values?
  - How are the boundaries of a data class isolated?
  - What data rates and data volume can the system tolerate?
  - What effect will specific combinations of data have on system operation?

- By applying black-box techniques, you derive a set of test cases that satisfy the following criteria:
  - test cases that reduce, by a count that is greater than one, the number of additional test cases that must be designed to achieve reasonable testing, and
  - test cases that tell you something about the presence or absence of classes of errors, rather than an error associated only with the specific test at hand.

Graph-Based Testing Methods

- Understand the objects in software
- Connect these objects using relationships.
- Define a series of tests - all objects have the expected relationship to one another.
- *A number of behavioral testing methods that can make use of graphs:*
  - Transaction flow modeling.
  - Finite state modeling.
  - Data flow modeling.
  - Timing modeling.

(a) Graph notation
(b) simple example

Equivalence Partitioning

- It is a black-box testing method.
- Divides the input domain into classes of data
- from these data - test cases can be derived.
- Test-case design - based on an evaluation of equivalence classes for an input condition.

Equivalence classes may be defined according to the following guidelines:

1. If an input condition ***specifies a range***, one valid and two invalid equivalence classes are defined.
2. If an input condition ***requires a specific value***, one valid and two invalid equivalence classes are defined.
3. If an input condition ***specifies a member of a set***, one valid and one invalid equivalence class are defined.
4. If an input condition ***is Boolean***, one valid and one invalid class are defined.

Boundary Value Analysis (BVA)

- More errors occurs at the boundaries in input
- Lesser errors in “center” in input
- For this reason that boundary value analysis (BVA) has been developed as a testing technique.
- BVA leads to a selection of test cases that exercise bounding values.

- **Guidelines for BVA are similar in many respects to those provided for equivalence partitioning:**
  - If an input condition **specifies a range bounded by values a and b,** test cases should be designed with values a and b and just above and just below a and b.
  - If an input condition **specifies a number of values**, test cases should be developed that exercise the minimum and maximum numbers. Values just above and below minimum and maximum are also tested.

- **Apply guidelines 1 and 2 to output conditions**. Test cases should be designed to create an output report that produces the maximum (and minimum) allowable number of table entries.
- If internal program data structures have prescribed boundaries (e.g., a table has a defined limit of 100 entries), be certain to design a test case to exercise the data structure at its boundary.

Orthogonal Array Testing

- Applied to problems in which the input domain is relatively small but too large to accommodate exhaustive testing.
- Useful in finding region faults - faulty logic within a software component.

Black-box Testing

- To find errors in the following categories:
  - incorrect or missing functions
  - interface errors
  - errors in data structures or external database access
  - behavior or performance errors
  - initialization and termination errors

Thank You

Session – 22

Test Driven Development

K L University
u/s 3 of UGC Act. 1956
Koneru Lakshmaiah Education Foundation

Test Driven Development (TDD) Vs. Agile Model Driven Development (AMDD)

TDD	AMDD
TDD shortens the programming feedback loop	AMDD shortens modeling feedback loop.
TDD is detailed specification	AMDD works for bigger issues
TDD promotes the development of high-quality code	AMDD promotes high-quality communication with stakeholders and developers.
TDD speaks to programmers	AMDD talks to business analyst, stakeholders, and data professionals.
TDD non-visually oriented	AMDD visually oriented
TDD has limited scope to software works	AMDD has a broad scope including stakeholders. It involves working towards a common understanding
Both support evolutionary development	----------

Examples of TDD:

Here in this example, we will define a class password. For this class, we will try to satisfy following conditions.

A condition for Password acceptance:

The password should be between 5 to 10 characters.

First, we write the code that fulfills all the above requirements.

```
package Prac;

import org.testng.Assert;
import org.testng.annotations.Test;

public class TestPassword {
  @Test
  public void TestPasswordLength() {
      PasswordValidator pv = new PasswordValidator();
      Assert.assertEquals(true, pv.isValid("Abc123"));

  }
}
```

Needed for TestNG

We can not run test because this class is not created yet

This is main validation test

Scenario 1:

To run the test, we create class PasswordValidator ();
We will run above class TestPassword ();

```
package Prac;

public class PasswordValidator {
 public boolean isValid(String Password)
 {
     if (Password.length()>=5 && Password.length()<=10)
     {
         return true;
     }
     else
         return false;
 }
}
```

This is main condition checking length of password. If meets return true otherwise false.

- Output is PASSED as shown below

```
<terminated> TestPassword [TestNG] C:\Program Files\Java\jre1.8.0_77\bin\javaw.exe (Jul 25, 2016, 2:10:22 PM)
[TestNG] Running:
  C:\Users\kanchan\AppData\Local\Temp\testng-eclipse--571370159\testng-customsuite.xml

PASSED: TestPasswordLength

===============================================
    Default test
    Tests run: 1, Failures: 0, Skips: 0
===============================================


===============================================
Default suite
Total tests run: 1, Failures: 0, Skips: 0
===============================================

[TestNG] Time taken by org.testng.reporters.EmailableReporter2@1b40d5f0: 202 ms
[TestNG] Time taken by org.testng.reporters.XMLReporter@28f67ac7: 63 ms
[TestNG] Time taken by org.testng.reporters.jq.Main@546a03af: 78 ms
[TestNG] Time taken by org.testng.reporters.JUnitReportReporter@5a01ccaa: 2 ms
[TestNG] Time taken by [FailedReporter passed=0 failed=0 skipped=0]: 1 ms
[TestNG] Time taken by org.testng.reporters.SuiteHTMLReporter@2b80d80f: 10 ms
```

Result of test as Passed

Advantages of TDD

- Early bug notification
- Better Designed, cleaner and more extensible code
- Confidence to Refactor
- Good for teamwork
- Good for Developers

Summary

- Test-driven development is a process of modifying the code in order to pass a test designed previously.
- It more emphasis on production code rather than test case design.
- In Software Engineering, It is sometimes known as "Test First Development."
- TDD includes refactoring a code i.e. changing/adding some amount of code to the existing code without affecting the behavior of the code.
- TDD when used, the code becomes clearer and simple to understand.

Questions

1. Distinguish Test Driven Development (TDD) Vs. Agile Model Driven Development (AMDD)
2. Explain any two scenarios of TDD with an Example
3. Distinguish TDD Vs. Traditional Testing
4. List out the advantages of TDD

Session – 24

CMMI and Six Sigma

K L University

u/s 3 of UGC Act. 1956

Koneru Lakshmaiah Education Foundation

Capability Maturity Model Integration

- Capability Maturity Model Integration (CMMI), a comprehensive process meta-model that is predicated on a set of system and software engineering capabilities that should be present as organizations reach different levels of process capability and maturity.
- The CMMI represents a process meta-model in two different ways: (1) as a “continuous” model and (2) as a “staged” model.

Levels of CMMI

Each process area (e.g., project planning or requirements management) is formally assessed against specific goals and practices and is rated according to the following capability levels:

- Level 0: Incomplete
- Level 1: Performed
- Level 2: Managed
- Level 3: Defined
- Level 4: Quantitatively managed
- Level 5: Optimized

The CMMI defines each process area in terms of “specific goals” and the “specific practices” required to achieve these goals. Specific goals establish the characteristics that must exist if the activities implied by a process area are to be effective. Specific practices refine a goal into a set of process-related activities.

CMMI Process Area Capability Profile

Specific Goals of CMMI

associated specific practices (SP) defined for project planning are:

- SG 1 Establish Estimates
  - SP 1.1-1 Estimate the Scope of the Project
  - SP 1.2-1 Establish Estimates of Work Product and Task Attributes
  - SP 1.3-1 Define Project Life Cycle
  - SP 1.4-1 Determine Estimates of Effort and Cost
- SG 2 Develop a Project Plan
  - SP 2.1-1 Establish the Budget and Schedule
  - SP 2.2-1 Identify Project Risks
  - SP 2.3-1 Plan for Data Management
  - SP 2.4-1 Plan for Project Resources
  - SP 2.5-1 Plan for Needed Knowledge and Skills
  - SP 2.6-1 Plan Stakeholder Involvement
  - SP 2.7-1 Establish the Project Plan
- SG 3 Obtain Commitment to the Plan
  - SP 3.1-1 Review Plans That Affect the Project
  - SP 3.2-1 Reconcile Work and Resource Levels
  - SP 3.3-1 Obtain Plan Commitment

CMMI also defines a set of five generic goals

The generic goals (GG) and practices (GP) for the project planning process area are:

- GG 1 Achieve Specific Goals
  - GP 1.1 Perform Base Practices
- GG 2 Institutionalize a Managed Process
  - GP 2.1 Establish an Organizational Policy
  - GP 2.2 Plan the Process
  - GP 2.3 Provide Resources
  - GP 2.4 Assign Responsibility
  - GP 2.5 Train People
  - GP 2.6 Manage Configurations
  - GP 2.7 Identify and Involve Relevant Stakeholders
  - GP 2.8 Monitor and Control the Process
  - GP 2.9 Objectively Evaluate Adherence
  - GP 2.10 Review Status with Higher-Level Management

- GG 3 Institutionalize a Defined Process
  - GP 3.1 Establish a Defined Process
  - GP 3.2 Collect Improvement Information
- GG 4 Institutionalize a Quantitatively Managed Process
  - GP 4.1 Establish Quantitative Objectives for the Process
  - GP 4.2 Stabilize Subprocess Performance
- GG 5 Institutionalize an Optimizing Process
  - GP 5.1 Ensure Continuous Process Improvement
  - GP 5.2 Correct Root Causes of Problems

Process area required to achieve a maturity Level

Level	Focus	Process Areas
Optimizing	*Continuous process improvement*	Organizational innovation and deployment Causal analysis and resolution
Quantitatively managed	*Quantitative management*	Organizational process performance Quantitative project management
Defined	*Process standardization*	Requirements development Technical solution Product integration Verification Validation Organizational process focus Organizational process definition Organizational training Integrated project management Integrated supplier management Risk management Decision analysis and resolution Organizational environment for integration Integrated teaming
Managed	*Basic project management*	Requirements management Project planning Project monitoring and control Supplier agreement management Measurement and analysis Process and product quality assurance Configuration management
Performed		

Six Sigma for Software Engineering

- Six Sigma is the most widely used strategy for statistical quality assurance in industry today.
- the Six Sigma strategy “is a rigorous and disciplined methodology that uses data and statistical analysis to measure and improve a company’s operational performance by identifying and eliminating defects’ in manufacturing and service-related processes”.
- The term Six Sigma is derived from six standard deviations—3.4 instances (defects) per million occurrences—implying an extremely high quality standard

What is Six Sigma

The Six Sigma methodology defines three core steps:

- Define customer requirements and deliverables and project goals via well defined methods of customer communication.
- Measure the existing process and its output to determine current quality performance (collect defect metrics).
- Analyze defect metrics and determine the vital few causes.

If an existing software process is in place, but improvement is required, Six Sigma suggests two additional steps:

- Improve the process by eliminating the root causes of defects.
- Control the process to ensure that future work does not reintroduce the causes of defects.
- These core and additional steps are sometimes referred to as the DMAIC (define, measure, analyze, improve, and control) method.

- **If an organization is developing a software process (rather than improving an existing process), the core steps are augmented as follows:**
- Design the process to (1) avoid the root causes of defects and (2) to meet customer requirements.
- Verify that the process model will, in fact, avoid defects and meet customer requirements.

This variation is sometimes called the DMADV (define, measure, analyze, design, and verify) method.

Questions

- Distinguish CMMI and Six Sigma Method.
- Explain various levels of CMMI in detail.
- Explain the core and additional steps of Six sigma methodology.
- List out specific goals and associated specific practices defined in CMMI.
- List out generic goals and practices of CMMI.

19CS2211 - Software Engineering

JUnit

Session 25

K L University

u/s 3 of UGC Act. 1956

Koneru Lakshmaiah Education Foundation

Outline

- Agenda:
  - Junit Architecture
  - Test case
  - Assert methods

History

- Kent Beck developed the first xUnit automated test tool for Smalltalk in mid-90's.
- Beck and Gamma (of design patterns Gang of Four) developed JUnit on a flight from Zurich to Washington, D.C.
- Junit has become the standard tool for Test-Driven Development in Java (see Junit.org)
- Junit test generators now part of many Java IDEs – Eclipse, BlueJ, Jbuilder, DrJava
- Xunit tools have since been developed for many other languages – Perl, C++, Python, Visual Basic, C#, …

Why create a test suite?

- Obviously you have to test your code—right?
  - You can do *ad hoc* testing (running whatever tests occur to you at the moment), or
  - You can build a test suite (a thorough set of tests that can be run at any time)
- **Disadvantages of a test suite**
  - It's a lot of extra programming
    - True, but use of a good test framework can help quite a bit
  - You don't have time to do all that extra work
    - *False!* Experiments repeatedly show that test suites reduce debugging time more than the amount spent building the test suite
- **Advantages of a test suite**
  - Reduces total number of bugs in delivered code
  - Makes code much more maintainable and re-factorable

Junit – Basic Structure

Junit - Detailed Architectural Overview

Architectural overview

- JUnit test framework is a package of classes that lets you write tests for each method, then easily run those tests
- **TestRunner** runs tests and reports **TestResult**s
- You test your class by extending abstract class ***TestCase***
- To write test cases, you need to know and understand the **Assert** class

Writing a TestCase

- To start using JUnit, create a subclass of TestCase, to which you add test methods
- Here's a skeletal test class:

```
import junit.framework.TestCase;
public class TestBowl extends TestCase {
} //Test my class Bowl
```

- Name of class is important – should be of the form TestMyClass or MyClassTest
- This naming convention lets TestRunner automatically find your test classes

Writing methods in TestCase

Pattern follows *programming by contract* paradigm:

- Set up **preconditions**
- Exercise functionality being tested
- Check **postconditions**

Example:

```
public void testEmptyList() {
    Bowl emptyBowl = new Bowl();
    assertEquals("Size of an empty list should be zero.",
                 0, emptyList.size());
    assertTrue("An empty bowl should report empty.",
    emptyBowl.isEmpty());
}
```

Things to notice:

- Specific method signature – public void ***test***Whatever()
  - Allows them to be found and collected automatically by JUnit
- Coding follows pattern
- Notice the assert-type calls…

Assert methods

- Each assert method has parameters like these: *message, expected-value, actual-value*
- Assert methods dealing with floating point numbers get an additional argument, a tolerance
- Each assert method has an equivalent version that does not take a message – however, this use is not recommended because:
  - messages helps documents the tests
  - messages provide additional information when reading failure logs

Assert methods Cont…

- assertTrue(String *message*, Boolean *test*)
- assertFalse(String *message*, Boolean *test*)
- assertNull(String *message*, Object *object*)
- assertNotNull(String *message*, Object *object*)
- assertEquals(String *message*, Object *expected*, Object *actual*) (uses equals method)
- assertSame(String *message*, Object *expected*, Object *actual*) (uses == operator)
- assertNotSame(String *message*, Object *expected*, Object *actual*)

More stuff in test classes

- Suppose you want to test a class Counter
- public class CounterTest
  extends junit.framework.TestCase {
  - This is the unit test for the Counter class
- public CounterTest() { } //Default constructor
- protected void setUp()
  - Test *fixture* creates and initializes instance variables, etc.
- protected void tearDown()
  - Releases any system resources used by the test fixture
- public void testIncrement(), public void testDecrement()
  - These methods contain tests for the Counter methods increment(), decrement(), etc.
  - Note capitalization convention

JUnit tests for Counter

```
public class CounterTest extends junit.framework.TestCase {
    Counter counter1;
    public CounterTest() { }   // default constructor

    protected void setUp() {   // creates a (simple) test fixture
        counter1 = new Counter();
    }

    public void testIncrement() {
        assertTrue(counter1.increment() == 1);
        assertTrue(counter1.increment() == 2);
     }

    public void testDecrement() {
        assertTrue(counter1.decrement() == -1);
    }
}
```

Note that each test begins with a *brand new* counter

This means you don't have to worry about the order in which the tests are run

TestSuites

- TestSuites collect a selection of tests to run them as a unit
- Collections automatically use TestSuites, however to specify the order in which tests are run, write your own:

```
public static Test suite() {
        suite.addTest(new TestBowl("testBowl"));
        suite.addTest(new TestBowl("testAdding"));
        return suite;
}
```

- Should seldom have to write your own TestSuites as each method in your TestCase should be independent of all others
- Can create TestSuites that test a whole package:

```
public static Test suite() {
        TestSuite suite = new TestSuite();
        suite.addTestSuite(TestBowl.class);
        suite.addTestSuite(TestFruit.class);
        return suite;   }
```

JUnit in Eclipse

- To create a test class, select File→ New→ Other… → Java, JUnit, TestCase and enter the name of the *class* you will test

Fill this in

This will be filled in *automatically*

Results

Your results are here

Java - TokenizerTest.java - Eclipse Platform

File Edit Source Refactor Navigate Search Project Run Window Help

JUnit (TokenizerTest)

Runs: 1 Errors: Failures:

Failures Hierarchy

testNext(TokenizerTest)

Failure Trace

junit.framework.AssertionFailedE
> but was:<1-three>
at TokenizerTest.testNext(Toke
at sun.reflect.NativeMethodAcc
at sun.reflect.NativeMethodAcc
at sun.reflect.DelegatingMetho
at junit.framework.TestResult$

Package Explo... Hierarchy JUnit

Tokenizer.java TokenizerTest.java

```
1  import junit.framework.TestCase;
2
3  /*
4   * Created on Jan 6, 2004
5   *
6   * To change the template for this generated file go t
7   * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;
8   */
9
10 import java.io.*;
11
12 /**
13  * @author dave
14  *
15  * To change the template for this generated type comme
16  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;
17  */
```

Outlir

Console [<terminated> C:\JBuilder9\jdk1.4\bin\javaw.exe (1/15/04 4:16 PM)]

```
n equals: expected = 3-=, found = 3-=
n equals: expected = 2-1, found = 2-1
```

Tasks Console

Unit testing for other languages

- Unit testing tools differentiate between:
  - Errors (unanticipated problems caught by exceptions)
  - Failures (anticipated problems checked with assertions)
- Basic unit of testing:
  - *CPPUNIT_ASSERT(Bool)* examines an expression
- CPPUnit has variety of test classes (e.g. *TestFixture*)
  - Inherit from them and overload methods

More Information

http://www.junit.org

Download of JUnit

Lots of information on using JUnit

http://sourceforge.net/projects/cppunit

C++ port of Junit

http://www.thecoadletter.com

Information on Test-Driven Development

Thank You

19CS2211 - Software Engineering

CMMI and Six Sigma

Session 26

K L University

u/s 3 of UGC Act. 1956

Koneru Lakshmaiah Education Foundation

Outline

- Agenda:
  - Definition – CMMI
  - Levels of CMMI
  - Six Sigma
  - Questions

CMMI

- Definition
  - The Capability Maturity Model Integration (CMMI) is a process and behavioral model that helps organizations streamline process improvement and encourage productive, efficient behaviors that decreases risks in software, product and service development.

Levels of CMMI

- Each process area (e.g., project planning or requirements management) is formally assessed against specific goals and practices and is rated according to the following capability levels:
- Level 0: Incomplete
  - Processes are viewed as unpredictable and reactive
  - an unpredictable environment that increases risk and inefficiency.

- Level 1: Performed
  - There's information on how to establish performance goals and then track those goals to make sure they're achieved at all levels of business maturity.
- Level 2: Managed
  - Projects - planned, performed, measured and controlled – at this level.
  - But there are still a lot of issues to address.
- Level 3: Defined
  - organizations are more proactive than reactive
  - A set of "organization-wide standards" to "provide guidance across projects, programs and portfolios."

- Level 4: Quantitatively managed
  - measured and controlled
  - quantitative data to determine predictable processes that align with stakeholder needs
- Level 5: Optimized
  - organization's processes are stable and flexible
  - organization will be in constant state of improving and responding to changes or other opportunities
  - Organization is stable, which allows for more "agility and innovation," in a predictable environment

CMMI Staged Maturity Levels

Peer Reviews Activities
Required for Level 3 and above

1. Prepare for Peer Reviews
2. Conduct Peer Reviews
3. Analyze Peer Review Data

Structure of CMMI - Process Areas
Maturity Level
Process Area 1
Process Area 2
Process Area N
Generic Goals
Specific Goals
Commitment
Ability
Directing Implementation
Verification
Features:
- Creating Policies
- Process Improvement
Features:
- Ensuring necessary resources
Features:
- Collect, measure & analyze that used to evaluate process
Features:
- Objective evaluation of confirmance
Specific Practices

- CMMI defines each process area in terms of specific goals
- Specific practices required to achieve these goals
- Specific practices refine a goal into a set of process-related activities

CMMI Process Area Capability Profile

Specific Goals of CMMI

- Associated specific practices (SP) defined for project planning are:
- SG 1 Establish Estimates
  - SP 1.1-1 Estimate the Scope of the Project
  - SP 1.2-1 Establish Estimates of Work Product and Task Attributes
  - SP 1.3-1 Define Project Life Cycle
  - SP 1.4-1 Determine Estimates of Effort and Cost
- SG 2 Develop a Project Plan
  - SP 2.1-1 Establish the Budget and Schedule
  - SP 2.2-1 Identify Project Risks
  - SP 2.3-1 Plan for Data Management
  - SP 2.4-1 Plan for Project Resources
  - SP 2.5-1 Plan for Needed Knowledge and Skills
  - SP 2.6-1 Plan Stakeholder Involvement
  - SP 2.7-1 Establish the Project Plan
- SG 3 Obtain Commitment to the Plan
  - SP 3.1-1 Review Plans That Affect the Project
  - SP 3.2-1 Reconcile Work and Resource Levels
  - SP 3.3-1 Obtain Plan Commitment

CMMI also defines a set of five generic goals

The generic goals (GG) and practices (GP) for the project planning process area are:

- GG 1 Achieve Specific Goals
  - GP 1.1 Perform Base Practices
- GG 2 Institutionalize a Managed Process
  - GP 2.1 Establish an Organizational Policy
  - GP 2.2 Plan the Process
  - GP 2.3 Provide Resources
  - GP 2.4 Assign Responsibility
  - GP 2.5 Train People
  - GP 2.6 Manage Configurations
  - GP 2.7 Identify and Involve Relevant Stakeholders
  - GP 2.8 Monitor and Control the Process
  - GP 2.9 Objectively Evaluate Adherence
  - GP 2.10 Review Status with Higher-Level Management

- GG 3 Institutionalize a Defined Process
  - GP 3.1 Establish a Defined Process
  - GP 3.2 Collect Improvement Information
- GG 4 Institutionalize a Quantitatively Managed Process
  - GP 4.1 Establish Quantitative Objectives for the Process
  - GP 4.2 Stabilize Subprocess Performance
- GG 5 Institutionalize an Optimizing Process
  - GP 5.1 Ensure Continuous Process Improvement
  - GP 5.2 Correct Root Causes of Problems

Process area required to achieve a maturity Level

Level	Focus	Process Areas
Optimizing	*Continuous process improvement*	Organizational innovation and deployment Causal analysis and resolution
Quantitatively managed	*Quantitative management*	Organizational process performance Quantitative project management
Defined	*Process standardization*	Requirements development Technical solution Product integration Verification Validation Organizational process focus Organizational process definition Organizational training Integrated project management Integrated supplier management Risk management Decision analysis and resolution Organizational environment for integration Integrated teaming
Managed	*Basic project management*	Requirements management Project planning Project monitoring and control Supplier agreement management Measurement and analysis Process and product quality assurance Configuration management
Performed		

Six Sigma for Software Engineering

- Six Sigma – suitable for statistical quality assurance in industry today
- It uses data and statistical analysis to measure and improve a company's operational performance by identifying and eliminating defects' in manufacturing and service-related processes
- The term Six Sigma is derived from six standard deviations—3.4 instances (defects) per million occurrences—implying an extremely high quality standard

What is Six Sigma

The Six Sigma methodology defines three core steps:

- Define
  - customer requirements
  - deliverables and
  - project goals

  via well defined methods of customer communication
- Measure the existing process and its output to determine current quality performance (collect defect metrics).
- Analyze defect metrics and determine the vital few causes.

If an existing software process is in place, but improvement is required, Six Sigma suggests two additional steps:

- Improve the process by eliminating the root causes of defects
- Control the process to ensure that future work does not reintroduce the causes of defects
- These core and additional steps are sometimes referred to as the DMAIC (define, measure, analyze, improve, and control) method

DEFINE
MEASURE
ANALYZE
IMPROVE
CONTROL

- If an organization is developing a software process (rather than improving an existing process), the core steps are augmented as follows:
  - Design the process
    - to avoid the root causes of defects
    - to meet customer requirements
  - Verify that the process model to avoid defects and meet customer requirements.

  This variation is sometimes called the DMADV (define, measure, analyze, design, and verify) method.

Sigma	Defects per 1,000,000
1σ	317,310
2σ	45,500
3σ	2,699
4σ	63
5σ	0.573
6σ	0.002

Questions

1. Distinguish CMMI and Six Sigma Method.
2. Explain various levels of CMMI in detail.
3. Explain the core and additional steps of Six sigma methodology.
4. List out specific goals and associated specific practices defined in CMMI.
5. List out generic goals and practices of CMMI.

Thank You