SOFTWARE PROCESS MODELS

Outline

- Software Development – Ideal vs. Reality
- Software Process and Process Models
- Code-and-Fix Model
- Waterfall Model
- Iterative and Incremental Models
  - Prototyping Model
  - Spiral Model
  - Rational Unified Process
  - Synchronize-and-Stabilize Model
  - Agile Models

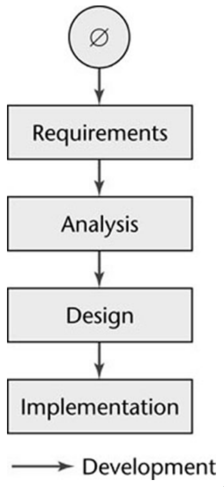
Software Process

- A <u>software process</u> is a framework for carrying out the activities of a project in an organized and disciplined manner.
  - Expresses the interrelationship among the phases of a software project by defining their order and frequency.
  - Defines the deliverables of the project.
- Specific software process implementations are called <u>software process models</u>.

Nandigam 3

Software Development – Ideal vs. Reality

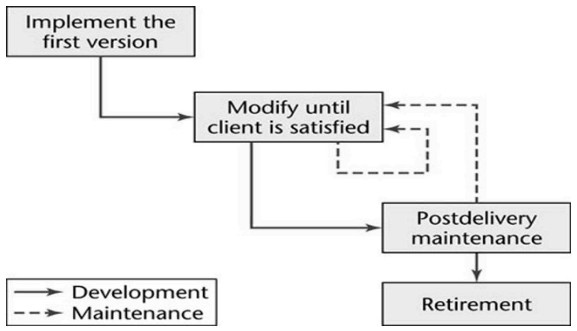
- Software development in an ideal world
  - Linear
  - Start from scratch
- In the real world, software development is considerably different
  - Software professionals make mistakes
  - The client's requirements change while the software product is being developed



Nandigam 4

Code-and-Fix Model

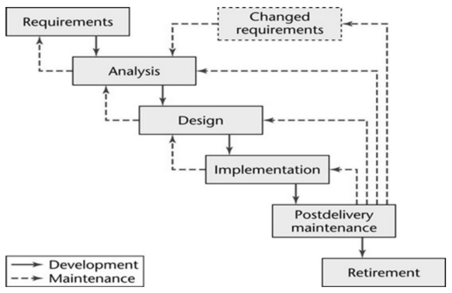
- No specifications
- No design
- Maintenance nightmare
- Easiest and worst way to develop software



Nandigam 5

Waterfall Model

- Known as classic life cycle (Winston Royce, 1970)
- Suggests a systematic, sequential approach.
- Document-driven – result of each phase is one or more documents that are approved.
- Big design up front
- Feedback loops generally restricted to adjacent phases.



Nandigam 6

Waterfall Model – Advantages

- Simple and easy to use with specific entrance and exit criteria for moving between phases.
- Easy to manage due to the rigidity of the model.
- Facilitates allocation of resources due to sequential nature of phases.
- Works well for smaller projects where requirements are very well understood.
- Documentation is produced at each phase.

Waterfall Model – Disadvantages

- Requirements must be known up front.
- No emphasis on active user feedback.
- No feedback of system by stakeholders until after testing phase – no intermediate versions.
- A working version of the program will not be available until late in the project time-span.
  - Customer must have patience.
  - Developers live in fear of this from customer - "I know this is what I asked for, but it isn't really what I wanted."
- Major problems with system aren't discovered until late in process.
- Inefficient use of resources – "blocking states" problem
- Hard to estimate reliably.

Iterative and Incremental Models

- The problem with the waterfall model is:
  "gather all the requirements, do all of the design, implement all of the code, and test all of the system in a linear fashion"
- Except for the smallest of projects this is impractical.
- Software is more naturally developed in a cyclical manner.
- Iterative Development is typified by repeated execution of the waterfall phases, in whole or in part, resulting in the refinement of requirements, design, and implementation.

Iterative and Incremental Models

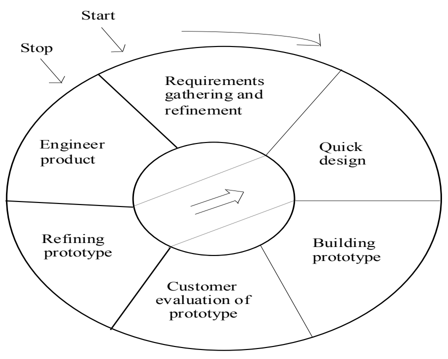
- An iterative process is incremental if each iteration
  - is relatively small.
  - delivers a piece of working code that supports a subset of final product functionality and features.
- Incremental development can be viewed as a "scheduling and staging strategy."
- User is closely involved in planning the next increment.
- Can be used to fight the "overfunctionality" syndrome.
  - Users find it to articulate real needs and tend to demand too much.
  - Focus on essential features first and additional functionality is only included if and when it is needed (and if the user is willing to pay for it ☺).

Iterative and Incremental Models

- Prototyping model
- Spiral model
- Synchronize-and-stabilize model
- Rational Unified Process
- Agile Models
  - XP, Scrum, Crystal, …

Prototyping Model

- A prototype is a working model of (part of) a final system.



Prototyping Paradigm

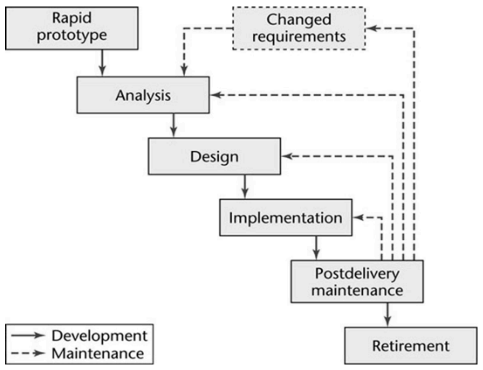
Prototyping Model

- Begin with requirements gathering – developer and customer meet and identify whatever requirements are known.
- A "quick-design" is then done.
- Build a prototype based on the quick design.
- Customer/user evaluates the prototype and requirements are refined.
- A process of iteration occurs as the prototype is "tuned" to satisfy the customer's requirements, while at the same time enabling the developer to better understand what needs to be done.
- After fully understanding the requirements, the prototype may be discarded (at least in part) and the actual software is engineered with an eye toward quality and maintainability.

Nandigam 13

Prototyping Model

- Prototyping is more commonly used as a technique that can be implemented within the context of a number of other process models.



Nandigam 14

Prototyping Model – Suitability

- Useful for eliciting customer requirements.
- Useful when the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that user interface should take.
- Useful as an important risk management technique.

Prototyping Model – Limitations

- Extensive prototyping can be expensive.
- When informed that the product must be rebuilt with requirements gained from prototyping, the customer cries foul and demands that "a few fixes" be applied to make the prototype a working product. (throwaway prototyping)
- The developer often makes implementation compromises in order to get a prototype working quickly and may forget or overlook to fix them when rebuilding the actual software. (evolutionary prototyping)
- The customer and developer must both agree and understand the role of a prototype in the software project development (throwaway or evolve?).

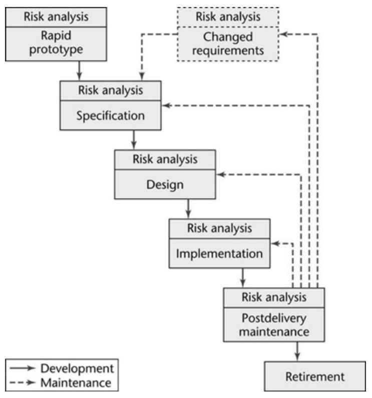
Spiral Model

- Software development is full of risks!
  - Project risks – affect the project planning
    - Resources are unavailable
    - Key personnel resign
    - Hardware manufacturer goes bankrupt
  - Technical risks – affect the actual project tasks
    - Personnel are untrained in a task
  - Business risks – affect the company building the software
    - Developing software that no one wants
    - Low-priced, functionally equivalent package by a competitor
- Minimizing risks via the use of prototypes, simulation, etc.

Nandigam 17

Spiral Model

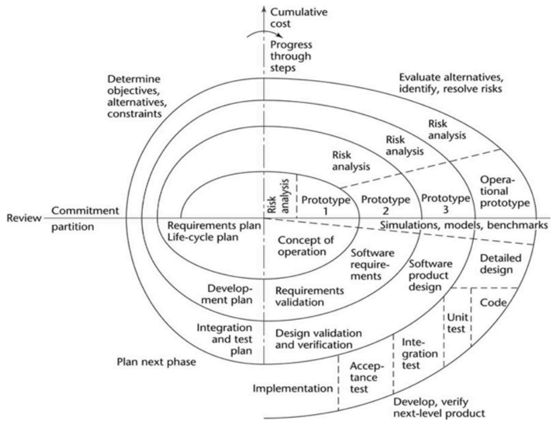
- Proposed by Barry Boehm
- Simplified form
  - Waterfall or Rapid prototyping model with risk analysis before each phase
  - Precede each phase by
    - Alternatives
    - Risk Analysis
  - Follow each phase by
    - Evaluation
    - Planning of next phase



Nandigam 18

Spiral Model

- Full Spiral model by Boehm



Nandigam 19

Spiral Model

- If risks cannot be resolved, then the project is immediately terminated or scaled down.
- Levels of risks
  - Catastrophic
  - Critical
  - Marginal
  - Negligible
- Need to prioritize risks
  - Top 20% of risks cause 80% of failures [Roger Pressman]

Nandigam 20

Spiral Model – Suitability

- Should be used for only large-scale software
  - May be overkill for small projects
  - Complicated to use – Risk analysis requires highly specific skills
- Generally used for in-house or non-commissioned software projects
  - When a project is terminated, software personnel can simply be reassigned to another project.
- Unlike in spiral model, in case of contract software, all risk analysis must be done before the contract is signed.
  - Breach-of-contract lawsuits

Synchronize-and-Stabilize Model

- Microsoft's software process model
- Requirements analysis
  - Interview potential customers
  - Extract a list of highest priority features
- Draw up specifications
- Divide project into 3 or 4 builds
  - First build – most critical features
  - Second build – next most critical features
  - …
- Each build is carried out by small teams working in parallel.

Synchronize-and-Stabilize Model

- At the end of the day — *synchronize* (test and debug)
- At the end of the build — *stabilize* (freeze the build)
- Repeated synchronization step ensures that various components always work together.
- Regular execution of the partially constructed product provides early insight into the operation of the product and allows modification of requirements if necessary during the course of a build.

Rational Unified Process (RUP)

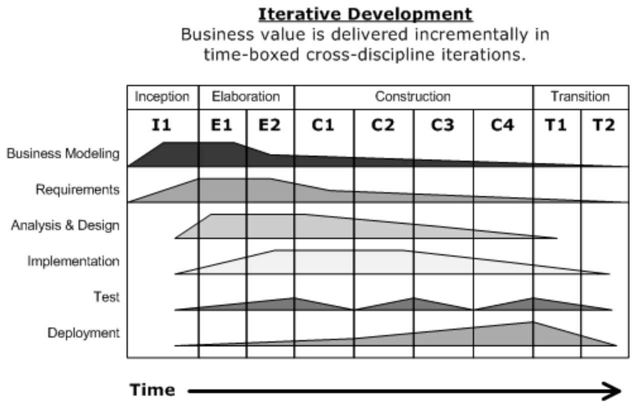
- Three of the most successful object-oriented methodologies/notations:
  - Booch Methodology by Grady Booch
  - Objectory Methodology by Ivar Jacobson
  - Object Modeling Technique by James Rumbaugh
- In 1999, Booch, Jacobson, and Rumbaugh unified their three separate methodologies into RUP.
  - Unified Software Development Process (USDP)
  - Unified Process (UP)
  - Rational Unified Process (RUP)
- Rational was bought by IBM in 2003.

RUP – Best Practices

- RUP captures six best practices
  - Develop iteratively
  - Manage requirements
  - Employ component-based architecture
  - Model software visually (with UML)
  - Continuously verify software quality
  - Control changes to software
- Heavyweight process (800-pound gorilla!!)
  - Originally conceived for large projects
  - May be overkill for small projects
- RUP is a software process product
  - sold by IBM like any other product
- IBM offers a RUP certification examination

RUP

- RUP is an iterative, use-case driven, and architecture-centric process.



RUP – Focus of Phases

- Inception Phase
  - Understanding requirements and defining scope.
- Elaboration Phase
  - Focus on requirements
  - Design and implementation to produce executable architectural prototype
  - Software architecture description
- Construction Phase
  - Focus on design and implementation
  - First operational product ("beta release")
- Transition Phase
  - Make sure the system has the right level of quality
  - Bug fixes, train users, adjust features, add missing elements
  - Deliver the final product ("public release")

RUP – Phase Milestones

- Each phase in RUP is concluded by a major milestone
  - Inception phase
    - Lifecycle objective (LCO)
  - Elaboration phase
    - Lifecycle architecture (LCA)
  - Construction phase
    - Initial operational capability (IOC)
    - first external release ("beta")
  - Transition
    - Product release (PR)

Agile Models

- Group of models based on iterative and incremental development.
- Lightweight software development methods.
- Alternative to document-driven, heavyweight software development methods.
- The Agile Manifesto reads as follows: (http://agilemanifesto.org/)

  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.

Agile Manifesto Principles

- The twelve guiding principles of Agile Manifesto (http://agilemanifesto.org/principles.html)
  1. Customer satisfaction by rapid delivery of useful software.
  2. Welcome changing requirements, even late in development.
  3. Working software is delivered frequently (weeks rather than months).
  4. Business people and developers must work together daily throughout the project.
  5. Projects are built around motivated individuals, who should be trusted.
  6. Face-to-face conversation is the best form of communication (co-location).

Agile Manifesto Principles

7. Working software is the principal measure of progress.
8. Sustainable development, able to maintain a constant pace (no overtime).
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.

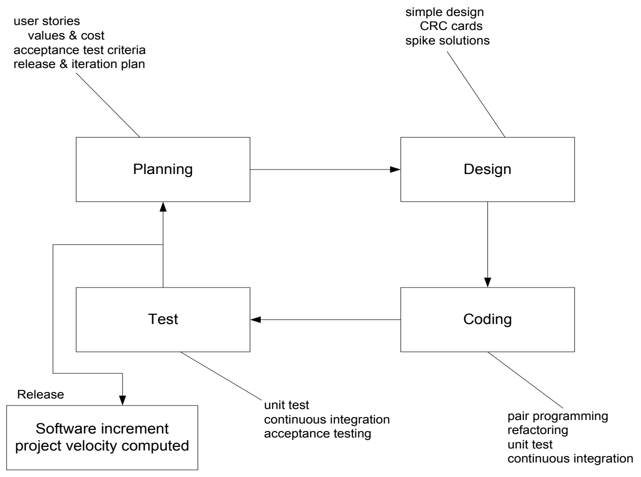
Nandigam 31

Agile Models – Examples

- Some example of agile models
  - Extreme Programming (XP)
  - Scrum
  - Crystal
  - Feature Driven Development (FDD)
  - Lean Software Development
  - Agile Unified Process

Nandigam 32

Extreme Programming (XP)

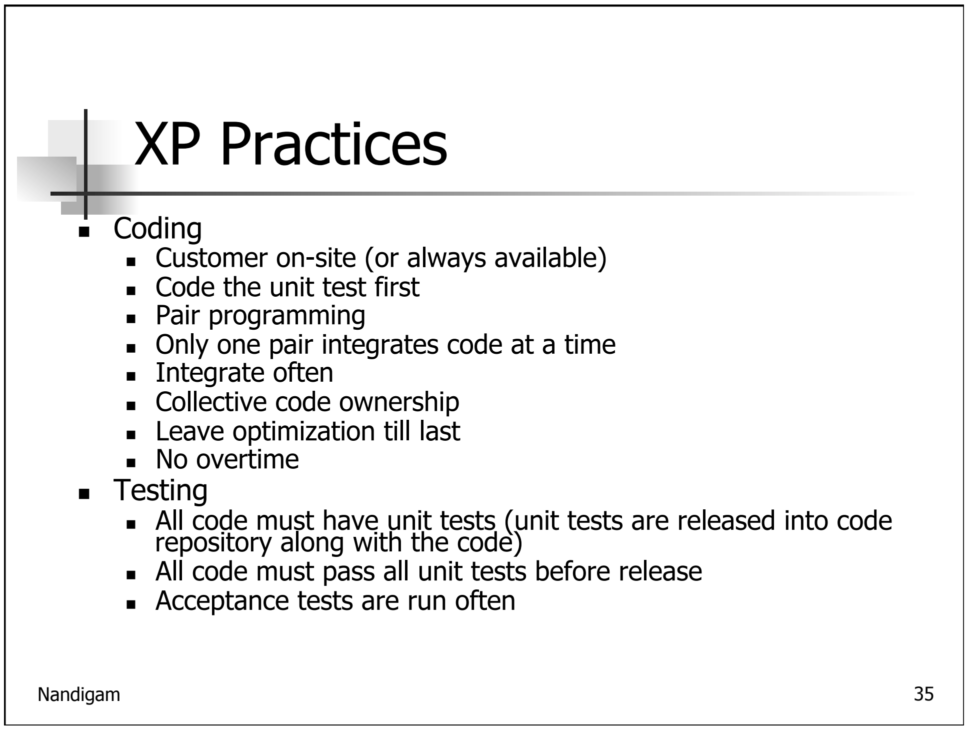


Nandigam 33

XP Practices

- Planning
  - User stories (written by customer or end user)
    - Things that the system needs to do for them
  - Release planning creates the schedule
  - Project is divided into iterations (iteration planning)
  - Make frequent small releases
  - Project velocity is computed after each release
  - Move people around
- Design
  - Simplicity
  - Use CRC cards for design sessions
  - Create spike solutions to reduce risk
  - No functionality is added early
  - Refactor whenever and wherever possible

Nandigam 34



XP Practices

- Coding
  - Customer on-site (or always available)
  - Code the unit test first
  - Pair programming
  - Only one pair integrates code at a time
  - Integrate often
  - Collective code ownership
  - Leave optimization till last
  - No overtime
- Testing
  - All code must have unit tests (unit tests are released into code repository along with the code)
  - All code must pass all unit tests before release
  - Acceptance tests are run often

Nandigam 35