E-notes Software Engineering

By Sagar N. Gharate

Syllabus

References :

1. Software Engineering : A Practitioner's Approach - Roger S. Pressman, McGraw hill(Eighth Edition) ISBN-13: 978-0-07-802212-8, ISBN-10: 0-07-802212-6
2. A Concise Introduction to Software Engineering - PankajJalote, Springer ISBN: 978-1-84800-301-9
3. The Unified Modeling Language Reference Manual - James Rambaugh, Ivar Jacobson, Grady Booch ISBN 0-201-30998-X

Chapter 2

Agile Development

Component Based Development

- Commercial off-the-shelf (COTS) software components, developed by vendors who offer them as products, provide targeted functionality with well-defined interfaces that enable the component to be integrated into the software that is to be built.
- The component-based development model incorporates many of the characteristics of the spiral model.
- It is evolutionary in nature, demanding an iterative approach to the creation of software.
- However, the component-based development model constructs applications from pre-packaged software components.

The component-based development model incorporates the following steps:

1. Available component-based products are researched and evaluated for the application domain in question.
2. Component integration issues are considered.
3. A software architecture is designed to accommodate the components.
4. Components are integrated into the architecture.
5. Comprehensive testing is conducted to ensure proper functionality.

- The component-based development model leads to software reuse, and reusability provides software engineers with a number of measurable benefits.

Agile Process Model:

- Agile SDLC model is a combination of iterative and incremental process models with focus on processed adaptability and customer satisfaction by rapid delivery of working software product.
- Agile Methods break the product into small incremental builds.
- Every iteration involves cross functional teams working simultaneously on various areas like planning, requirements analysis, design, coding, unit testing, and acceptance testing.

Advantages:

- Customer satisfaction by rapid, continuous delivery of useful software.
- Customers, developers and testers constantly interact with each other.
- Close, daily cooperation between business people and developers.
- Continuous attention to technical excellence and good design.
- Regular adaptation to changing circumstances.
- Even late changes in requirements are welcomed.

Disadvantages:

- In case of some software, it is difficult to assess the effort required at the beginning of the software development life cycle.
- There is lack of emphasis on necessary designing and documentation.
- The project can easily get taken off track if the customer representative is not clear what final outcome that they want.

- Only senior programmers are capable of taking the kind of decisions required during the development process.

Product and Process:

- If the process is weak, the end product will undoubtedly suffer.
- But a compulsive overreliance on process is also dangerous.
- People derive as much (or more) satisfaction from the creative process as they do from the end product.
- An artist enjoys the brush strokes as much as the framed result.
- A writer enjoys the search for the proper metaphor as much as the finished book.
- As creative software professional, you should also derive as much satisfaction from the process as the end product.
- The duality of product and process is one important element in keeping creative people engaged as software engineering continues to evolve.

Agility and Agile Process Model:

Agility:

- Agility is dynamic, content specific, aggressively change embracing, and growth oriented
- It encourages team structures and attitudes that make communication (among team members, between technologists and business people, between software engineers and their managers) more simplistic.
- It emphasizes rapid delivery of operational software and de-emphasizes the importance of intermediate work products.
- It recognizes that planning in an uncertain world has its limits and that a project plan must be flexible.
- Agility can be applied to any software process.

Agile Process:

- Any agile software process is characterized in a manner that addresses a number of key assumptions

1. It is difficult to predict in advance which software requirements will persist and which will change. It is equally difficult to predict how customer priorities will change as the project proceeds.
2. For many types of software, design and construction are interleaved. That is, both activities should be performed in tandem so that design models are proven as they are created. It is difficult to predict how much design is necessary before construction is used to prove the design.
3. Analysis, design, construction, and testing are not as predictable as we might like.

Agility Principles:

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.
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.
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 behaviour accordingly.

Agile Process Models:

- Extreme Programming (XP)
- Adaptive Software Development (ASD)
- Dynamic Systems Development Method (DSDM)
- Scrum
- Crystal
- Feature Driven Development (FDD)
- Agile Modelling (AM)

- **Human Factors -**

- “Agile development focuses on the talents and skills of individuals, moulding the process to specific people and teams.” i.e. The process molds to the needs of the people and team, not the other way around.
- The key traits that must exist among the people on an agile team are:
    - Competence.
    - Common focus.
    - Collaboration.
    - Decision-making ability.
    - Fuzzy problem-solving ability.
    - Mutual trust and respect.
    - Self-organization

Extreme Programming (XP) :

What is Extreme Programming?

XP is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop a software.

- Extreme Programming (XP), the most widely used approach to agile software development proposed by Kent Beck.
- Recently (now.date.31Aug2022), a variant of XP, called Industrial XP (IXP) has been proposed which refines XP and targets the agile process specifically for use within large organizations.

XP Process :

Extreme Programming uses an object-oriented approach that encompasses a set of rules and practices that occur within the context of four framework activities:

- Planning
- Design
- Coding
- Testing

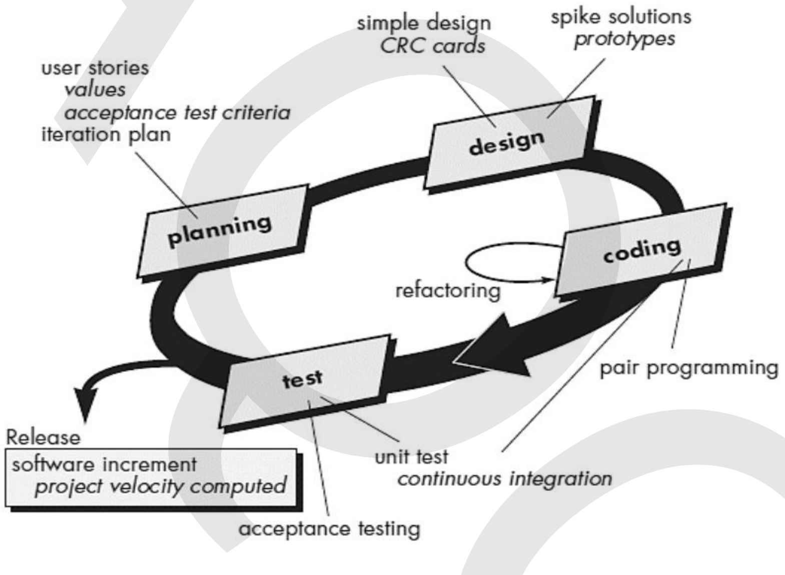

Fig. Extreme Programming Process

Planning:

- Begins with the creation of a set of stories (also called user stories)
- Each story is written by the customer and is placed on an index card
- The customer assigns a value (i.e. a priority) to the story
- 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.

Design:

- Follows the keep it simple principle
- Encourage the use of CRC (class-responsibility-collaborator) cards
- For difficult design problems, suggests the creation of "spike solutions"—a design prototype
- Encourages "refactoring"—an iterative refinement of the internal program design
- Design occurs both before and after coding commences.

Coding:

- Recommends the construction of a series of unit tests for each of the stories before coding commences.
- Encourages "pair programming"
    - Developers work in pairs, checking each other's work and providing the support to always do a good job.
    - Mechanism for real-time problem solving and real-time quality assurance
    - Keeps the developers focused on the problem at hand
- Needs continuous integration with other portions (stories) of the s/w, which provides a "smoke testing" environment.

Testing:

- Unit tests should be implemented using a framework to make testing automated. This encourages a regression testing strategy.
- Integration and validation testing can occur on a daily basis
- Acceptance tests, also called customer tests, are specified by the customer and executed to assess customer visible functionality.
- Acceptance tests are derived from user stories

Extreme Programming - Values :

XP sets out to lower the cost of change by introducing basic values, principles and practices. By applying XP, a system development project should be more flexible with respect to changes.

Extreme Programming Values

Extreme Programming (XP) is based on the five values :

- Communication
- Simplicity
- Feedback
- Courage
- Respect

- **Communication**

Communication plays a major role in the success of a project. Problems with projects often arise due to lack of communication. Many circumstances may lead to the breakdown in communication. Some of the common problems are

- A developer may not tell someone else about a critical change in the design.
- A developer may not ask the customer the right questions, and so a critical domain decision is blown.
- A manager may not ask a developer the right question, and project progress is misreported.
- A developer may ignore something important conveyed by the customer.

Extreme Programming emphasizes continuous and constant communication among the team members, managers and the customer. The Extreme Programming practices, such as unit testing, pair programming, simple designs, common metaphors, collective ownership and customer feedback focus on the value of communication.

XP employs a coach whose job is to notice when the people are not communicating and reintroduce them. Face-to-Face communication is preferred and is achieved with pair programming and a customer representative is always onsite.

- **Simplicity**

Extreme Programming believes in ‘it is better to do a simple thing today and pay a little more tomorrow to change it’ than ‘to do a more complicated thing today that may never be used anyway’.

- Do what is needed and asked for, but no more.
  - ''Do the simplest thing that could possibly work'' The DTSTTCPW principle.
  - Implement a new capability in the simplest possible way. Also known as the KISS principle ‘Keep It Simple, Stupid!’.
  - A coach may say DTSTTCPW when he sees an Extreme Programming developer doing something needlessly complicated.
  - Refactor the system to be the simplest possible code with the current feature set. This will maximize the value created for the investment made till date.
- Take small simple steps to your goal and mitigate failures as they happen.
- Create something that you are proud of and maintain it for a long term for reasonable costs.
- Never implement a feature you do not need now i.e. the ‘You Aren’t Going to Need It’ (YAGNI) principle. Communication and Simplicity support each other.
- The more you communicate the clearer you can see exactly what needs to be done, and you gain more confidence about what really need not be done.
- The simpler your system is, the less you have to communicate about the fewer developers that you require. This leads to better communication.

- **Feedback**

Every iteration commitment is taken seriously by delivering a working software. The software is delivered early to the customer and a feedback is taken so that necessary changes can be made if needed. Concrete feedback about the current state of the system is priceless. The value of the feedback is a continuously running system that delivers information about itself in a reliable way.

In Extreme Programming, feedback is ensured at all levels at different time scales -

- Customers tell the developers what features they are interested in so that the developers can focus only on those features.
- Unit tests tell the developers the status of the system.

- The system and the code provides feedback on the state of development to the managers, stakeholders and the customers.
- Frequent releases enable the customer to perform acceptance tests and provide feedback and developers to work based on that feedback.
- When the customers write new features/user stories, the developers estimate the time required to deliver the changes, to set the expectations with the customer and managers.

Thus, in Extreme Programming the feedback-

- Works as a catalyst for change
- Indicates progress
- Gives confidence to the developers that they are on the right track.

- **Courage**

Extreme Programming provides courage to the developers in the following way-

- To focus on only what is required
- To communicate and accept feedback
- To tell the truth about progress and estimates
- To refactor the code
- To adapt to changes whenever they happen
- To throw the code away (prototypes)

This is possible as no one is working alone and the coach guides the team continuously.

- **Respect**

Respect is a deep value, one that lies below the surface of the other four values. In Extreme Programming,

- Everyone respects each other as a valued team member.
- Everyone contributes value such as enthusiasm.
- Developers respect the expertise of the customers and vice versa.
- Management respects the right of the developers to accept the responsibility and receive authority over their own work.

Combined with communication, simplicity, and concrete feedback, courage becomes extremely valuable.

- Communication supports courage because it opens the possibility for more high-risk, high-reward experiments.
- Simplicity supports courage because you can afford to be much more courageous with a simple system. You are much less likely to break it unknowingly.
- Courage supports simplicity because as soon as you see the possibility of simplifying the system you try it.
- Concrete feedback supports courage because you feel much safer trying radical modifications to the code, if you can see the tests turn green at the end. If any of the tests do not turn green, you know that you can throw the code away.

Industrial XP

IXP incorporates six new practices that are designed to help ensure that an XP project works successfully for significant projects within a large organization.

1. **Readiness assessment:**
   The assessment ascertains whether (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 organizational culture will support the new values of an agile team, and (5) the broader project community will be populated appropriately.
2. **Project community:**
   Classic XP suggests that the right people be used to populate the agile team to ensure success. The people on the team must be well-trained, adaptable and skilled, and have the proper temperament to contribute to a self-organizing team.
3. **Project chartering:**
   Chartering also examines the context of the project to determine how it complements, extends, or replaces existing systems or processes.
4. **Test-driven management:**
   Test-driven management establishes a series of measurable “destinations “and then defines mechanisms for determining whether or not these destinations have been reached.
5. **Retrospectives:**
   An IXP team conducts a specialized technical review after a software increment is delivered. Called a retrospective, the review examines “issues, events, and lessonslearned” across a software increment and/or the entire software release.
6. **Continuous learning:**
   Because learning is a vital part of continuous process improvement, members of the XP team are encouraged to learn new methods and techniques that can lead to a higher quality product.

Adaptive Software Development (ASD):

Adaptive Software Development (ASD) is a technique for building complex software and systems. ASD focus on human collaboration and team self-organization.

ASD incorporates three phases Speculation, Collaboration, and Learning.

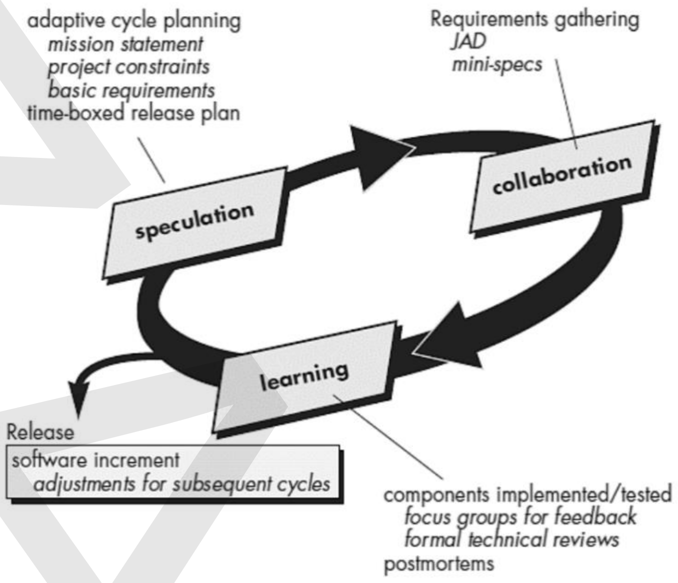

Fig. Adaptive Software Development

Speculation:

- "Speculate" refers to the planning paradox—outcomes are unpredictable, therefore, endless suppositions on a product's look and feel are not likely to lead to any business value.
- The big idea behind speculate is when we plan a product to its smallest detail as in a requirements up front Waterfall variant, we produce the product we intend and not the product the customer needs.
- In the ASD mindset, planning is to speculation as intention is to need.

Collaboration:

- Collaboration represents a balance between managing the doing and creating and maintaining the collaborative environment."
- Speculation says we can't predict outcomes. If we can't predict outcomes, we can't plan. If we can't plan, traditional project management theory suffers.
- Collaboration weights speculation in that a project manager plans the work between the predictable parts of the environment and adapts to the uncertainties of various factors—stakeholders, requirements, software vendors, technology, etc.

Learning:

- "Learning" cycles challenge all stakeholders and project team members.
- Based on short iterations of design, build and testing, knowledge accumulates from the small mistakes we make due to false assumptions, poorly stated or ambiguous requirements or misunderstanding the stakeholders' needs.

- Correcting those mistakes through shared learning cycles leads to greater positive experience and eventual mastery of the problem domain.

Dynamic Systems Development Methods (DSDM):

- The Dynamic Systems Development Method is an agile software development approach that “provides a framework for building and maintaining systems which meet tight time constraints through the use of incremental prototyping in a controlled project environment”
- DSDM is an iterative software process in which each iteration follows the 80 percent rule.
- That is, only enough work is required for each increment to facilitate movement to the next increment.
- The remaining detail can be completed later when more business requirements are known or changes have been requested and accommodated.

DSDM life cycle that defines three different iterative cycles, preceded by two additional life cycle activities:

Feasibility study- establishes the basic business requirements and constraints associated with the application to be built and then assesses whether the application is a viable candidate for the DSDM process.

Business study- establishes the functional and information requirements that will allow the application to provide business value; also, defines the basic application architecture and identifies the maintainability requirements for the application.

Functional model iteration- produces a set of incremental prototypes that demonstrate functionality for the customer.

Design and build iteration- revisits prototypes built during functional model iteration to ensure that each has been engineered in a manner that will enable it to provide operational business value for end users.

Implementation- places the latest software increment into the operational environment.

- DSDM can be combined with XP to provide a combination approach that defines a solid process model (the DSDM life cycle) with the nuts and bolts practices (XP) that are required to build software increments.
- In addition, the ASD concepts of collaboration and self-organizing teams can be adapted to a combined process model.

Scrum:

- Scrum principles are consistent with the agile manifesto and are used to guide development activities within a process that incorporates the five framework activities: requirements, analysis, design, evolution, and delivery.
- Within each framework activity, work tasks occur within a process pattern called a sprint.
- The work conducted within a sprint (the number of sprints required for each framework activity will vary depending on product complexity and size) is adapted to the problem at hand and is defined and often modified in real time by the Scrum team.
- Scrum emphasizes the use of a set of software process patterns that have proven effective for projects with tight timelines, changing requirements, and business criticality.
- Each of these process patterns defines a set of development actions: Backlog—a prioritized list of project requirements or features that provide business value for the customer.
- Items can be added to the backlog at any time (this is how changes are introduced).
- The product manager assesses the backlog and updates priorities as required.

Agile Unified Process (AUP):

Agile Unified Process (AUP) is a simplified version of the Rational Unified Process (RUP) developed by Scott Ambler. It describes a simple, easy to understand approach to developing business application software using agile techniques and concepts yet still remaining true to the RUP. The AUP applies agile techniques including test-driven development (TDD), agile modeling (AM), agile change management, and database refactoring to improve productivity.

It distinguishes Development Release iterations from Production Release iterations. It works on building up a model, implementing, testing and deploying it and then followed by configuration management. The focus is on the covering the high-value activities rather than covering every possible thing

Unlike the RUP, the AUP has only seven disciplines:

1. **Model**- Understand the business of the organization, the problem domain being addressed by the project, and identify a viable solution to address the problem domain.
2. **Implementation** - Transform model(s) into executable code and perform a basic level of testing, in particular unit testing.
3. **Test** - Perform an objective evaluation to ensure quality. This includes finding defects, verifying that the system works as designed, and validating that the requirements are met.
4. **Deployment** - Plan for the delivery of the system and to execute the plan to make the system available to end users.
5. **Configuration Management** - Manage access to project artifacts. This includes not only tracking artifact versions over time but also controlling and managing changes to them.
6. **Project Management**- Direct the activities that take place within the project. This includes managing risks, directing people (assigning tasks, tracking progress, etc.), and coordinating with people and systems outside the scope of the project to be sure that it is delivered on time and within budget.

7. **Environment** - Support the rest of the effort by ensuring that the proper process, guidance (standards and guidelines), and tools (hardware, software, etc.) are available for the team as needed.

The Agile Unified Process is based on the following philosophies:

1. Your staff knows what they're doing. People are not going to read detailed process documentation, but they will want some high-level guidance and/or training from time to time. The AUP product provides links to many of the details, if you are interested, but doesn't force them upon you.
2. Simplicity. Everything is described concisely using a handful of pages, not thousands of them.
3. Agility. The Agile UP conforms to the values and principles of the agile software development and the Agile Alliance.
4. Focus on high-value activities. The focus is on the activities which actually count, not every possible thing that could happen to you on a project.
5. Tool independence. You can use any toolset that you want with the Agile UP. The recommendation is that you use the tools which are best suited for the job, which are often simple tools.
6. You'll want to tailor the AUP to meet your own needs.

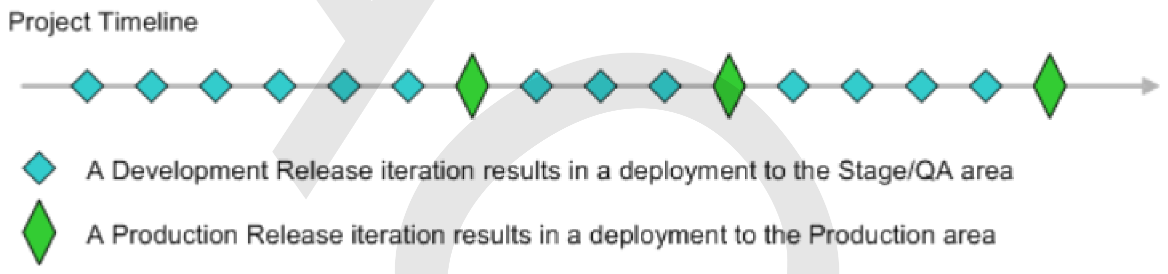
Project Timeline
A Development Release iteration results in a deployment to the Stage/QA area
A Production Release iteration results in a deployment to the Production area

Fig. Project Timeline

The Agile Unified Process distinguishes between two types of iterations. A development release iteration results in a deployment to the quality-assurance and/or demo area. A production release iteration results in a deployment to the production area. This is a significant refinement to the Rational Unified Process.