Software Engineering
Agile Software Development (Chp#03)

Agile vs Plan-Driven:
Agile development approach emphasized flexibility, collaboration, and incremental delivery. It prioritize adaptability to changing requirements and feedback and rapid deliveries. Moreover minimal documentation and focus on working code. Program specification, design and implementation are inter-leaved.

Plan-Drive Development based on separate development stages with the output produced at end of each stages planned in advanced. Moreover required proper documentation and very detailed specification and design before implementation.

Agile Methods: Aims to reduce overhead in the software process, limiting documentation and quick responsive.
The principle of agile methods are:

1. Customer Involvement: customer should involve throughout the development so that evaluation and feedback can achieve
2. Incremental delivery: work on small increment which enable customer specifying the requirement to be include in iteration.
3. People not process: team member should left to develop their own ways of working without prescriptive process.
4. Change Adaptable: design system to later accommodate change easily.

Agile Method Applicability:

1. Product development where developing small or medium product for sale.
2. Startup and Entrepreneurship: agile allows to quickly develop and iterate on their products in a dynamic environment.
3. Project where there is clear commitment from customer to become involve in development process.

Agile Method techniques:

1. **Extreme Programming (XP):** it take an extreme approach to iterative development.
   The principle of Extreme Programming are:
   - **Incremental Planning**: Developer break the stories into development task or make small chunks.
   - **Small Release:** release of system are frequent and incrementally add functionality to first release
   - **Pair Programming/ people not process**: developer work in pairs, checking each other's work and providing support to always do a good job. It encourages refactoring as whole team effort for improving system code.
   - **Refactoring**: all developer are expected to refactor the code continuously as soon any improvement are found. This helps to keep the code simple and maintainable. Programming team look for possible software improvement and make improvement. Changes are easier to make when the code is well-structured and clear.
     Example:
     - Re-organization of class hierarchy to remove duplicate code.
   - **Test-first Development**: Testing approach where program is tested after every changes has been made. Writing test code clarifies the requirements to be implemented. Include user involvement in test development and validation.
     Problem with Test-first Development:
     - They may write incomplete tests that do not check all possible exception that may occur
     - Some test can be very difficult to write incrementally in complex user interface.
   - **Customer involvement**: customer is member of dev team and is responsible for brining system requirements to the team for implementation.

Agile project Management: the principal responsibility of software project managers is to manage the project so that the software is deliver on time and within the planned budget for the project.

2. **Scrum:** is an agile method that focuses on managing iterative development rather than specific agile practices. There are three phases in Scrum:
    - Initially outline planning phase where we establish objective of project and design software architect
    - Followed by series of sprint cycles. Where each cycle develops an increment of system
    - Wrap up the project, completes required documentation. And assesses the lesson learned from the project

    Scrum Benefits:
    - Product is broken down into set of understandable and manageable chunks
    - Whole team have visibility of everything and consequently team communication is improved.
    - Customer see on-time delivery of increments and gain feedback on how the product works.
    - Trust between customer and developers is establish which result in project to succeed.

    Some Terminologies:
    - Sprint: A development iteration, usually 2-3 weeks long.
    - Product Backlog: list of work to be done on the project
    - Scrum: short face-to-face meeting that includes whole team that reviews progress and prioritize work.
    - Scrum Master: ensuring that process is followed and guides the team in effective use of scrum. It also arranges daily meeting, tracks the backlog of work to be done, and also measures progress.
    - Velocity: how much product backlog effort that team can cover in a single sprint.
    - Development team: Self-organizing group of developers, not more than 7 people.

    Complete Sprint Cycle: cycle is normally 2-4 weeks. Firstly planning is for the product backlog. Then selection phase involves customer to select features and functionality from product backlog. Once both customer and developer agreed, the team is isolated from customer and only communicate through scrum master. The role of scrum master to protect development team from external distraction. At end of sprint, the work done is reviewed and presented to stakeholders, and then the sprint cycle another begins.

3. Scaling Agile Method: it been know agile work efficient in small or medium size project but Scaling Up agile method involves changing these to cope with larger, longer projects, where there are multiple development teams, perhaps working in different locations.
    - Scaling Up is concern using agile method for developing large system that can't be develop by small team
    - Scaling Out concern with how agile method can be introduced across a large organization.

Problems with Agile Method:
- Incompatible with legal approach to contracts that is commonly used in large companies
- Agile method are not efficient for software maintenance.
- Agile method designed for small co-located team. Yet now development involve worldwide distributed team.
- Contractual Issues: A contract that pays for developer time rather than functionality required.

Agile method and software Maintenance: Many organization spend more on maintenance, if agile method are successful, they have to support maintenance as well as original development. Moreover Problem may arise if original development team cannot be maintained.
Two Key issues:
- Are system developed by agile method are maintainable because of lack of documentation for future developer.
- In large organization, where strict legal contracts definition followed. Thus agile method effective to customer change request in large organization.

Agile Maintenance: Agile development relies on dev team and it will not always work on the system the same developer. Key problems are:

- Lack of product documentation
- Keeping customer involvement in development process
- Maintain the continuity of the development team

Agile and Plan-based Factors:

1. System Issues:
    - How large system is: As agile method are most effective for small co-located team project.
    - Type of system: system with lot of analysis need a detailed design to carry out.
    - System lifetime: long-lifetime systems require documentation to communicate and support team

2. People and teams:
    - Technologies: visualization and program analysis is essential if design documentation is not available
    - How Dev team organized: Design documentation may require if team is distributed.
    - Competence: Agile method required higher skills programmer and designers to code and implement.

3. Organizational Issues:
    - Contract: In large organization follows legal frameworks for contact definition.
    - Delivery: Will customer be available to provide feedback of system increment?
    - Culture: can informed agile development fir into organizational culture of detailed documentation?

Agile Method for Large Systems: Agile method is been challenging when applied to large system, which often consist of separate development team. With lengthy procurement and development cycles, maintaining cohesive team over time becomes challenging. Thus, while agile emphasized flexibility and incremental development, their application to large system should deal with all difficulties and limitation.
Factors is large System: Brownfield Development, diverse stakeholder, system configuration, regulatory constraints.

Scaling up to large system:

- A completely incremental approach to requirements engineering is impossible
- There cannot be single product owner of customer representative.
- For large systems development, it is not possible to focus only on code of the system
- Cross-team communication mechanisms have to be designed and used.

Multi-team Scrum:

- Role Replication: Each team has product owner for their work component and scrum Master.
- Product architect: each team choses a product architect which collaborate with other architect to design.
- Release alignment: the date of product release from each team are aligned
- Scrums of scrum: a daily scrum of scrums where each team representative discuss the progress and plan

Requirements Engineering (Chp #04):

What is Requirement Engineering?
The system requirements are the descriptions of the system services and constraints that are generated during the requirements engineering process.

What is Requirement?
It may range from high level abstract statement of service or system to detailed mathematical functional specification.

Types of Requirement:
User requirements: Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.

System requirements: A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.

Stake Holders: Any person or organization who is affected by system and has legitimated interst. Can be end user, system owner and managers.

Agile methods and requirements:
Agile methods usually use incremental requirements engineering and may express requirements as 'user stories'. This is practical for business systems but problematic for system that requires pre-delivery analysis or system developed by several team.

Functional and non-functional requirements:
Functional: "What the system should do" define the specific behaviors and functions of the system outlining its capability and features. It is user oriented. Example: authentication, data validation, calculation, data processing,

Non-functional: "How the system should perform". It specify qualities or attributes that system mush haves, it is system oriented. Example: Secure, user interaction, responsiveness, performance, scalability, constraints, If these are not met, the system may be useless.

Non-Functional Classification:
1. Product Requirement: delivered product must behave in a particular way that is speed, reliability.
2. Organizational Requirement: Policy and procedures that is implementation requirements.
3. External Requirements: Arise from factors which hare external to the system

Requirement Engineering Process:
1. **Requirements Elicitation**: involves gathering requirements from stakeholders, who may include end-users, customer, and domain experts. Involve system performance, hardware constraints. Stage includes:
    - Requirement discovery requirements classification, requirements negotiation, requirement specification
        - **Problem of requirement elicitation** is that stake holder don't know what they really want. Moreover different stake holder have conflicting requirements.

Ethnography: Ethnography is effective for understanding existing processes but cannot identify new features that should be added to a system.
- **The problem with ethnography** is that it studies existing practices which may have some historical basis which is no longer relevant.

Stories and Scenarios:
Scenarios and user stories are real-life examples of how a system can be used. Stories and scenarios are a description of how a system may be used for a particular task. Because they are based on a practical situation, stakeholders can relate to them and can comment on their situation with respect to the story.

Scenarios: A structure form of user story. Should include:

- A description of the starting situation.
- A description of the normal flow of events;
- A description of what can go wrong;
- Information about other concurrent activities;
- A description of the state when the scenario finishes.

2. **Requirement Specification**: The process of writing down the user and system requirements in a requirements document. The specification should be understandable to stakeholders and serve as basis for development and testing activities.
   Ways of writing a system requirements specification:
   - **Natural language**: Written in everyday language with numbered sentences each expressing one requirement
   - **Structured natural language**: Follows a standard form or template. But problem is the lack of clarity.
   - **Design description languages**: Uses language similar to programming but more abstract.
   - **Graphical notations**: Utilizes graphical models like UML diagrams.
   - **Mathematical specifications**: Based on mathematical concepts such as finite-state machines or sets.

   ⇨ **Form Based Specification:**
   - Definition of the function or entity.
   - Description of inputs and where they come from.
   - Description of outputs and where they go to.
   - Information about the information needed for the computation and other entities used.
   - Description of the action to be taken.
   - Pre and post conditions (if appropriate).
   - The side effects (if any) of the function.

   ⇨ **Tabular based specification**: table form of representation where one column mapping to other. Usually columns are condition and other is the action.

3. **Requirement Validation**: Requirements error costs are high so validation is very important Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.
   Requirements checking:
   - **Validity**: Ensuring the system offers functions that align well with the customer's needs.
   - **Consistency**: Checking for conflicts or contradictions among requirements.
   - **Completeness**: Verifying that all functions requested by the customer are included.
   - **Realism**: Assessing whether the requirements can be implemented within the available budget and technology constraints.
   - **Verifiability**: Confirming that the requirements can be checked or tested to ensure they have been met.

   Requirements validation techniques:
   - Requirements reviews: Systematic manual analysis of the requirements.
   - Prototyping: Using an executable model of the system to check requirements.
   - Test-case generation: Developing tests for requirements to check testability.

Review Checks:

- **Verifiability**: is the requirement realistically testable?
- **Traceability**: is the origin of requirement clearly stated?
- **Adaptability**: can the requirement be changes without large impact on other requirements.

4. **Requirement Change:** The business and technical environment of the system always changes after installation due to new hardware, market changes and software updates. There are diverse user which have their own demand, so to fulfill their need is the big challenging in big projects.

Requirements management: is like keeping track of a to-do list that constantly changes. As a project progresses, new tasks might pop up, and some old ones might change. It's important to keep everything organized, make sure tasks are connected if they depend on each other, and have a way for suggesting and implementing changes smoothly.

Requirement management planning:

- **Requirements identification**: Each requirement must be uniquely identified for cross-referencing.
- **A change management process**: Assessing impact and cost of changes.
- **Traceability policies**: Defining relationships between requirements and system design.
- **Tool support**: Utilizing tools from specialist systems to spreadsheets for managing requirements.

Requirements change management: Deciding if requirements change should be accepted

- **Problem analysis and change specification**: Analyzing the validity of proposed changes.
- **Change analysis and costing**: Assessing the impact of proposed changes and making decisions based on traceability information.
- **Change implementation:** Modifying the requirements document and system design accordingly.