**Chapter 3 and 4 (up to 99) Notes**

**Chapter 3**

Today’s businesses require rapid development which models that plan completely before developing cannot satisfy.

Rapid SW Development processes are designed to produce useful SW quickly. The SW is not developed as a single unit but as a series of increments, each increment including new functionality.

**Fundamental Characteristics of Rapid-Development Processes**

1. The processes of specification, design, and implementation are interleaved
   1. No detailed system specification…design documentation is minimalized or generated automatically…Only user requirements document defines the most important characteristics of the system
2. System is developed in series of versions
   1. End-users and stakeholders are involved in specifying and evaluating each version
3. System UI are often developed using an interactive development system for quick design.

Agile Methods are incremental development methods in which increments are small and typically available every 2-3 weeks for the customer

**Agile Methods**

1. Types of Development where Agile Development is Successful
   1. Small to medium sized projects
   2. Custom system development within an organization
      1. Where there is clear commitment from the customer
      2. Where there are not a lot external rules and regulations that affect SW
2. Principles of Agile Methods
   1. Customer Involvement
      1. They should be closely involved throughout the process
   2. Incremental Delivery
   3. People not process
      1. Team member should be left to develop their own ways of working without prescriptive processes
   4. Embrace change
      1. Expect system requirements to change…so design the system to acommadate these changes
   5. Maintain simplicity
3. Difficulties with the Principles of Agile Methods
   1. Success depends on a willing and committed customer
   2. Team member smay not have suitable personalities for the instennse involvement that is typical of agile methods
   3. Prioritizing changes can be extremely difficult
   4. Maintaining simplicity requires extra work
   5. Difficult for plan-oriented companies to move towards an incremental, and informal model
4. 2 Question to consider when Considering Agile Methods and Maintanence
   1. Are systems that are developd using an agile approach maintainable, given the emphasis in the development process of minimizing formal documention?
   2. Can Agile methods be used effectively for evolving a system in response to customer change request?
5. Difficulty arises in the Maintenance phase because it is difficult to keep the customer involved. Also team mebers must rely on understanding aspects of the system without consulting documentation (this is much worse for separated teams).

**Plan-driven and Agile Development (Section 3.2)**

1. Differences
   1. Plan-driven
      1. Idetnfiies separate stages and has each stage have associated outputs
      2. Formal documents are used to communicate between stages
   2. Agile
      1. Considers Design and implementation to be central activities…the other activities are incorporated into these activities
2. Most projects blend Plan-driven and Agile methods
3. Questions to ask to decide on blend of Plan-driven and Agile methods
   1. Is it important to have a very detailed specification and design bfore moving to implementation?
   2. Is an incremental delivery strategy realistic?
   3. How large is the system being developed?
   4. What type of system is being developed?
   5. What is the expected system lifetime?
   6. What technologies are available to support system development?
   7. How is the development team organized?
   8. Are there cultural issues that may affect the system development? (Company is more Plan-driven orientted)
   9. How good are the designers and programmers in the development team?
   10. Is the system subject to external regulation?

**Extreme Programming (Section 3.3)**

1. Programmers work in pairs and develop test for each task before writing code
   1. All of the test must be successful before code is integrated into the system
2. Requirements are expressed as scranrios (called User Stories)
3. Release Cycle
   1. Select User Stories for this Release
   2. Break Down Stories to Tasks
   3. Plan Release
   4. Develop/Integrate/Test SW
   5. Release SW
   6. Evaluate System
   7. Go back to step a (Select User Stories)
4. Practices
   1. Incremental development is supported through small, frequent releases…requirements are based on User Stories
   2. Customer involvement is supported through interaction between customer and development team
   3. People are supported through pair programming, collective ownership of system code, and sustainable development process that does NOT involve long working hours
   4. Change is embraced
   5. Maintaining simplicity by constant refactoring
5. More XP Practives
   1. Incremental Planning
   2. Small Releases
   3. Simple Design
   4. Test-first development
   5. Refactoring
   6. Pair Programming
   7. Collective Ownership
   8. Continuous integration
   9. Sustainable pace
   10. On-site customer
6. XP address the Incremental Issue of system degrading after many changes by constantly forcing the refactoring of code

**Testing in XP (Section 3.3.1)**

1. Key features of testing gin XP
   1. Test-first development
   2. Incremental test development from scenarios
   3. User involvement in the test development and validation
      1. Customer creates acceptance tests for the stories that are to be implemented in the next release
   4. The use of automated testing frameworks
2. Reasons why testing may NOT be thorough
   1. Programmers prefer programming to testing and therefore, may take shortcuts when writing tests
   2. Some tests can be very difficult to write incrementally
   3. It is difficult to judge the completeness of a set of tesets

**Pair Programming (Section 3.3.2)**

1. Programmers work in pairs to develop SW
   1. They actually sit together to develop software
   2. However, the same pairs do not always program together…pairs are dynamically create so that all team members work with each other during the development process
2. Advantages
   1. Promotes team collective responsibility…individuals are not held responsible for problems within the code
   2. Acts as an informal review process because each line of code is looked at by at least 2 people…it helps speed up code review
   3. It helps support refactoring
3. People think pair programming is much slower than solo programming
   1. But it is comparable in time (when looking at students) because
      1. Pairs discuss SW before development so they have less false starts and less rework
      2. Errors are avoided thanks to the informal inspection 🡪 less time is spend on debugging
   2. The results do not hold though for experienced programmers….there is a significant loss of productivity

**Agile Project Management – Scrum Model (Section 3.4)**

1. Does not use pair programming or test-first development
2. 3 Phases in Scrum
   1. Outline Planning and Architecture Design Phase
      1. General objectives for the project and design the SW architecture
   2. Sprint Cycles (this can be repeated)
      1. Each cycle develops an increment of the system
      2. Cycles
         1. Asses
         2. Select
         3. Develop
         4. Review
   3. Project Closure
      1. Completes the required documentation, assesses lessons learned from the project, and wraps up the project
3. Key Characteristics
   1. Sprints are fixed length
      1. Usually 2-4 weeks
      2. Correspond to development or a release in XP
   2. Starting point for planning is the product backlog (list of work to be done on the project)
      1. During the Assess phase of the Sprint Cycle, this is reviewed, and priorities and risks are assigned.
      2. Customer is closely involved in this process and can introduce new requirements or tasks at the beginning of each sprint
   3. Selection phase in Sprint Cycle involves all of the project team who work with the customer to select the features and functionality to be developed during the Sprint
   4. Once these are agreed, the team organizes themselves to develop SW
      1. Short daily meetings involving all team members are held to review progress and maybe reprioritize work
   5. At the end of the sprint cycle, the work done is reviewed and present to stakeholders. The next cycle then begins
4. Advantages
   1. Project is broken down in a set of manageable and understandable chunks
   2. Unstable requirements do not hold up progress
   3. The whole team has visibility of everything and consequently team communication is improved
   4. Customer see on-time delivery of increments and gain feedback on how the product works
   5. Trust between customer and developers is established and positive culture is created in which everyone expects the project to succeed

**Scaling Agile Methods (Section 3.5)**

1. Original Agile Methods were developed for use by small programming teams
   1. The need for large systems to be developed quickly has increase the need for Agile Methods to be scaled up for use with large systems
2. Differences between Large and Small System Development
   1. Large systems are usually collections of separate, communicating systems, where separate team develop each system
      1. It is practically impossible for each team to have a view of the whole system
   2. Large systems include and interact with a number of existing systems
      1. Many system requirements are concerned with this interaction and so don’t really lend themselves to flexibility and increment development
   3. Where several systems are integrated to create a system, a significant fraction of the development is concerned with system configuration rather than original code development
   4. Large systems and their development processes are often constrained by external rules and regulations limiting the way that they can be developed, that require creating types of system documentation to be produced, etc.
   5. Large systems have a long procurement and development time
   6. Large systems usually have a diverse set of stakeholders (e.g. Nurses and administrators may be the end-users of a medical system but senior medical staff, hospital managers, etc. are also stakeholders)
3. 2 Perspectives on Scaling Agile Methods
   1. Scaling Up
      1. Concerned with using these methods for developing large system that cannot be developed by a small team
   2. Scaling out
      1. Concerned with how Agile methods can be introduced across a large organization with many years of SW development experience
4. Agile Methods must adapt to be used for large systems
   1. It is not possible to focus only on the code of the system
      1. You need to do more up-front design and system documentation
      2. System architecture has to be designed and there has to be documentation produced to describe critical aspects of the system
   2. Cross-team communication mechanisms have to be designed and used
   3. Continuous integration, where the whole system is built every tie any developer check s in a change, is practically impossible when several separate programs have to be integrated to create the system
      1. Essential to maintain frequent system builds and regular releases of the system.
5. Difficulties for Large Companies to use Agile Methods
   1. Project managers who do not have experience of agile methods may be reluctant to accept the risk of a new approach
   2. Large organization often has quality procedures and standards that all projects are expected to follow and, because of their bureaucratic nature, these are likely to be incompatible with agile methods.
   3. Agile methods seem to work best when team members have a relatively high skill level.
      1. However, in large companies, there are likely to be a wide range of skills and abilities
   4. There may be cultural resistance to agile methods, especially in those organizations that have a long history of using conventional systems engineering processes.

**Key Points**

1. Agile methods are incremental development methods that focus on rapid development, frequent release of the SW, reducing process overheads, and producing high-quality code
   1. They involve the customer directly
2. The decision on whether to an agile or a plan-driven approach to development should depend on the type of SW being developed, capabilities of the development team, and the culture of the company
3. Extreme Programming is an Agile method that uses frequent releases of the SW, continuous software improvement, and customer participation in the development team
4. A particular strength of extreme programming is the development of automated test before a program feature is created (All test must be successful prior to integration)
5. Scrum method is an Agile Method that provides a project management framework. It is centered around a set of sprints, which are fixed time periods when a system increment is developed. Planning is based on prioritizing a backlog of work and selecting the highest-priority tasks for a sprint
6. Scaling Agile methods for large system is difficult.
   1. Large systems need up-front design and some documentation
   2. Continuous integration is practically impossible when there are several separate development team working on a project

**Chapter 4**

**User Requirements**: Statements, in a natural language plus diagrams, of what services the system is expected to provide to system users and the constraints under which it must operate

* These tend to be general
* Readers are NOT usually concerned with how the system will be implemented

**System Requirements**: More detailed descriptions of the SW system’s functions, services, and operational constraints. This document (aka functional specification) should define exactly what is to be implemented.

* These tend to be more specific
* Readers need to know more precisely what the system will do

**Functional and Non-Functional Requirement (Section 4.1)**

1. System Requirements (not User) are classified either as Functional or Non-Functional
2. Functional:
   1. Statements of services the system should provide
   2. How the system should react to particular inputs
   3. How the system should behave in particular situation
   4. Maybe, state what the system should NOT do
3. Non-Functional:
   1. Constraints on the services or functions offered by the system
   2. Timing constraints
   3. Constraints on development process
   4. Constraints imposed by standards

**Functional Requirements (Section 4.1.1)**

1. Describe what the system should do
2. Define specific facilities to be provided by the system (these have been taken from the User Requirements document)
3. Should be COMPLETE and CONSISTENT
   1. Completeness: All services required by the user should be define
   2. Consistency: Requirements should not have contradictory definitions

**Non-Functional Requirements (Section 4.1.2)**

1. Requirements that are not directly concerned with the specific services delivered by the system to its users
2. Usually deal with performance, security, availability, reliability, response time, and store occupancy requirements/constraints.
3. It is difficult to relate components to Non-functional requirements (it is easier to relate component to functional requirements) because:
   1. Non-functional requirements may affect the overall architecture of a system rather than individual components (i.e. Performance)
   2. A single non-functional component, such as security requirement, may generate a number of related functional requirements
4. Types of Non-Functional Requirements
   1. Product Requirements
      1. Specify or constrain the behavior of the software (Ex. Performance requirements, reliability requirements, security requirements, and usability requirements)
   2. Organizational Requirements
      1. Broad system requirements derived from policies and procedures in the customer’s and developer’s organization. (Ex. Operational process requirements, development process requirements, and environmental process requirements)
   3. External Requirements
      1. All requirements that are derived from factors external to the system and its development process (Ex. Regulatory requirements, legislative requirements, and ethical requirements)
5. Metrics for Specifying Non-Functional Requirements
   1. You should write Non-Functional Requirements quantitatively so that they can objectively tested
   2. Metrics
      1. Speed
         1. Processed transactions/second, Response time, Screen refresh time
      2. Size
         1. Mbytes, # of ROM chips
      3. Ease of Use
         1. Training time, Number of help frames
      4. Reliability
         1. Mean time to failure, Probability of unavailability, rate of failure occurrence, and availability
      5. Robustness
         1. Time to restart after failure, % of events causing failure, probability of data corruption on failure
      6. Portability
         1. % of target dependent statements, # of target systems
   3. Non-Functional requirements often conflict with other Non-Functional and Functional Requirements.

**The SW Requirements Document (Section 4.2)**

1. AKA SRS (SW Requirements Specification)
   1. Includes both the User Requirements and Detailed Specification of the System Requirements
      1. Agile Methods argue that the moment SRS is compiled, it is quickly out of date so there is no point to do it at all
2. Users of a Requirements Document
   1. System Customers
      1. Specify the requirements check that they meet their need. Specify changes to the requirements
   2. Managers
      1. Use the requirements document to plan a bid for the system to plan the system development process
   3. System Engineers
      1. Use the requirements to understand what system is to be developed
   4. System Test Engineers
      1. Use the requirements to develop validation tests
   5. System Maintenance Engineers
      1. Use the requirements to understand the system and the relationships between components
3. Structure of Requirements Document
   1. Preface
      1. Define the readership of the document, version history of the document, and summary of changes of each version
   2. Introduction
      1. Describe the need for the system. Describe the function, and explain how it will work with other systems.
   3. Glossary
      1. Define technical terms
   4. User Requirement Definition
      1. Describe the services provided for the user. Non-functional Requirements should be described here. Product and process standards that must be followed should be specified
   5. System Architecture
      1. High-level overview of the anticipated system architecture
   6. System Requirements Specification
      1. Describe Functional and Non-Functional requirements in more detail
   7. System Models
      1. Graphical models showing relationships between system components, the system, and its environment.
   8. System Evolution
      1. Fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on.
   9. Appendices
      1. Specific information related to the application (ex. Hardware and Database specifications)
   10. Index

**Requirements Specification (Section 4.3)**

1. Is the process of writing down the User and System Requirements in a requirements document
2. User Requirements should describe the Functional and Non-Functional Requirements sot that they are understandable by system users who don’t have detailed technological knowledge.
3. System Requirements are expanded version of the User Requirements that are used by SW Engineers as the starting point for the system design.
   1. They should not be concerned with how the system should be designed or implemented
   2. It is practically impossible to exclude all design information because:
      1. You may have to design an initial architecture of the system to help structure the requirements specification
      2. System must interoperate with existing systems, which constrain the design and impose requirements on the new system
      3. The use of specific architecture to satisfy Non-Functional Requirements may be necessary.
4. Writing Requirements Specifications
   1. User
      1. Written in natural language with appropriate diagrams and tables
   2. System
      1. Natural Language sentences
         1. Written using numbered sentences each expressing one requirement
      2. Structured natural language
         1. Written in natural language on a standard form or template. Each field provides info about an aspect of the requirement
      3. Design description language
         1. Uses language like a programming language but with more abstract features to specify the requirements by defining an operational model of the system.
      4. Graphical notations
         1. Graphical models like UML
      5. Mathematical specifications
         1. Based on mathematical concepts such as finite state machines

**Natural Language Specification (Section 4.3.1)**

1. Guidelines
   1. Invent a standard format and ensure that all requirement definitions adhere to it
   2. Use language consistently to distinguish between mandatory and desirable requirements
   3. Use text highlighting to pick out key parts of the requirement
   4. Do not assume that readers understand technical SWE language
   5. Try to associate a rationale with each user requirement

**Structured Specifications (Section 4.3.2)**

1. The freedom of the requirements writer is limited and all requirements are written in a standard way
2. Use templates to specify system requirements andmay use programming language constructs to show alternatives, iteration and may highlight key elements using shading or different fonts
3. When a standard form is used for requirements, include the following info:
   1. Description of the function or entity being specified
   2. Description of its inputs and where these come from
   3. Description of ites outputs and where these go to
   4. Info about the info that is needed for the computation or other entities in the ystem that are used
   5. A description of the action to be taken
   6. If a functional approach is used, a pre-condition and a post-condition
   7. Description of the side effects (if any) of the operation