CSC-584 Assignment-1

By Neil Duraiswami   
SID: 212620514

**Contents**

[**Chapter 1:** Introduction 2](#_Toc158035654)

[**Chapter 2:** Software Processes 6](#_Toc158035655)

[**Chapter 3:** Agile Software Development 9](#_Toc158035656)

[**References:** 13](#_Toc158035657)

# **Chapter 1: Introduction**

1.  
A screenshot of a software development program

Description automatically generated

In most cases, generic software products do not meet the specific requirements and preferences of individual users or organizations even though they may have been specified by their developer. Consequently, in line with the capabilities programmed into the system, new workflows must be developed by those initiating them to address these user limitations and process inefficiencies that may arise. The software does not allow for this type of customization; thus, it makes things difficult for users such as those with specific needs who will likely have a bad experience using it altogether.

2.  
**The four important attributes are:**

**Maintainability:** The professional software should be developed to be progressive and responsive toward future requirements. This calls for the codebase of the software to be well-structured and documented such that it becomes easy for developers to understand, modify, or even extend or update it without having any chances of slipping bugs into the system.

**Dependability and Security:** Reliability, security, and safety are all part of dependability. Reliable software should work the same way every time it is used without making mistakes. Security ensures that the software will be invulnerable to unauthorized access or malicious activities, thereby ensuring it protects itself as well as its users from any damage.

**Efficiency:** The efficient software utilizes system resources in a way that does not waste memory, processor cycles, and other related resources. Thus, this also involves concerns about responsiveness, processing time, and memory utilization hence ensuring the software performs its tasks promptly with no unnecessary delays or inefficiencies.

**Acceptability:** Acceptable software is designed to take into consideration the people who are expected to use it thus making such systems self-explanatory, user-friendly, and compatibility with other interacting systems. User-friendly interfaces, clean documentation, and running on existing workflow contribute much to the overall acceptability of software leading to user satisfaction and adoption.

**The four additional attributes that may be significant in professional software are:**

**Scalability:** Scalability is the ability of a program to manage increased workloads or growth in user base without loss of performance and functionality. A software system that scales well should be able to adapt to changing demands by employing resources effectively and spreading tasks among several parts or servers.

**Flexibility:** Flexibility deals with how easily software can be altered or customized to meet new requirements or changes in business processes. A flexible software system allows for modular design, configuration options, and extensibility, allowing developers to add new features without affecting the existing functionality too much.

**Interoperability:** Interoperability means that software can communicate and exchange data smoothly with other systems or platforms. An interoperable software system builds on conforming to industry standards and protocols thus making integration of third-party applications or services possible. This empowers interoperable software for its smooth operation in heterogeneous environments and ecosystems.

**Usability:** Usability is all about the ease of use and accessibility of software for the end-users. Intuitive interfaces, clear navigation, and informative feedback are some characteristics of a usable software system thereby enabling the users to interact with the software efficiently. Usability testing and user feedback are important in identifying and addressing usability problems to enhance overall user experience.

3.

Specialized software engineering techniques are required by different application types because of their unique characteristics and requirements. For instance:

**Entertainment Systems:** Iterative design processes and rapid prototyping are necessitated by an emphasis on user experience and interactivity.

Example: Video game development is characterized by iterative cycles of designing, testing, and refining that enhance gameplay experiences for the players.

**The Modeling and Simulation Systems:** High-performance computing resources coupled with specialized algorithms are essential to handle computationally intensive tasks as well as complex simulations.

Example: Meteorological forecasting systems use supercomputers capable of simulating atmospheric conditions using complex mathematical models thus giving accurate predictions about future weather patterns.

**Data Collection Systems:** Interaction with sensors, operation in hostile environments, robust error-handling mechanisms, and fault-tolerant designs become necessary.

Example: Environmental monitoring systems deployed in remote areas should be able to withstand harsh environmental conditions and intermittent connectivity problems while collecting data from various sensors.

**Systems of Systems:** A system of systems is a reason various dissimilar systems should have uniform interfaces and protocols.

For example, enterprise resource planning (ERP) systems combine financial, human resources, and supply chain automation software modules that require standardized application programming interfaces (APIs) or data formats to enable communication between different sub-systems.

# **Chapter 2: Software Processes**

4.

A diagram of a diagram

Description automatically generated

When the development process begins, capturing the initial requirements for the system to be developed is what "Requirements Specification" is all about. The requirements are built on stakeholder needs and expectations and provide a basis for other subsequent stages in the SDLC. This phase ensures that there is a clear understanding of the desired functionality and goals of the software at the start. “Requirements Modification” comes after the component analysis stage whereby existing components are identified for reuse. In addition to this, since reused components may not perfectly align with an initially specified requirement, there may be a need for their integration into systems with some modifications done to them. During this phase, it reflects on initial requirements based on which adjustments or refinements will then be made to allow the reuse of these components. The nonexistence of exact matches may necessitate revising certain requirements to fit in well with available components’ abilities or exploring other possible solutions.

The initial requirements engineering activity is meant to establish that the original requirements capture the needs and wants of stakeholders, which are crucial in developing a basis for the developmental process. However, the subsequent “Requirements Modification” activity may be needed to modify these requirements based on the reuse components identified. This is aimed at ensuring that system functionality fits with available component capabilities. Using existing parts often necessitates modifying system requirements so that they correspond to specific functionalities offered by such components. When there are separate requirements modification activities, however, it enables developers to systematically scrutinize them thus optimizing reuse benefits of software and handling any mismatches or disagreements.

5.

Software development’s dynamic nature and many factors make change inevitable within complex systems. Initially, there is the need to adjust software systems because of changing user expectations over time. These changes often increase when users interact with software and gain an understanding of the modifications. Additionally, market forces, regulations, and technology shifts require software systems to adapt to remain competitive and compliant. Furthermore, organizational restructuring may result from mergers or acquisitions that cause corresponding changes in business procedures necessitating corresponding changes in software systems for corporate alignment purposes. In general, the contemporary software system’s complexity alongside its ever-changing environment makes it certain that a developer must incorporate flexibility into any development process if he wants to succeed in his work.

In addition to prototyping and incremental delivery, several software process activities from Boehm’s spiral model assist in predicting changes and improving the robustness of the software under construction:

**A diagram of a diagram

Description automatically generated**

**Objective Setting:** Defining set objectives for each phase of the project can help identify beforehand potential areas that could undergo some changes. Teams can anticipate changes on their way to achieving these goals by establishing explicit targets and constraints such as performance levels or regulatory requirements.

**Risk Assessment and Reduction:** Performing comprehensive risk analysis enables team members to identify possible sources of change and uncertainty in a project. For example, risks associated with technological dependencies, market instability, or changing user needs can be identified enabling proactive planning to mitigate these risks. Some strategies include developing contingency plans or adopting modular designs which would minimize the effects expected from those changes.

**Development and Validation:** The software can be made more resilient to change by choosing the right development model, based on identified risks and project constraints. For example, adopting an iterative or agile development approach enables frequent feedback for changing needs. Continuous testing and stakeholder feedback help in validating software so that it aligns with changing expectations and can be modified accordingly.

**Planning:** Regularly assessing a project’s progress and modifying plans according to assumed conclusions enables teams to foresee changes as well as respond wisely to them. Teams should proactively modify their development strategies based on emerging trends and incorporate feedback from stakeholders in addressing a shift in the needs of customers or alterations in market environments.

6.

**Software Specification:** The initial phase is concerned with defining the limits and requirements of software to be developed. It involves collecting and documenting functional and non-functional requirements such as attributes, performance, usability, and security. The software specification serves as a road map in the whole development process that shows what should follow. For an all-inclusive understanding of their needs and desires, it is important to involve stakeholders like clients as well as end users.

**Software Design and Implementation:** After setting up the software specifications, the design and implementation phase begins. In this stage, software developers transform the conceptual design into a tangible form. Based on specified requirements, software architects together with developers work hand in hand to develop architectures for software, data structures algorithms user interfaces among other things. The development process includes writing codes, testing individual components as well as integrating them into one complete system. Design patterns are used conforming to coding standards to guarantee the reliability, maintenance supportability, or scalability of the software system.

**Software Validation:** After the implementation of software, this must pass through rigorous testing to ensure that its functioning is proper. The validation phase is meant to affirm that the software meets specified requirements and behaves expectedly under different circumstances. Various methods are used to find and get rid of any bugs or faults in the system during its development stages such as unit testing, integration testing, system testing, and acceptance testing. Peer reviews and automated testing tools are some of the quality assurance processes used for maintaining software quality and reliability. The validation phase continues iteratively until the software meets the predefined quality standards and is ready for deployment.

**Software Evolution:** Software is not static. It evolves to meet changing needs, technological advancements, and user feedback. Software evolution comprises maintenance, updates, or enhancements to existing software. Change management processes include managing change requests which entail bug fixes among others, performance optimization, and feature enhancement but with minimal disruption to the system’s stability and functionality. Version control systems help keep track of changes hence ensuring integrity of software throughout its lifecycle. Agile practices are like continuous improvement.

# **Chapter 3: Agile Software Development**

7.

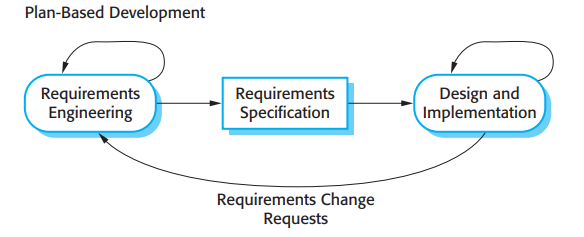
To accelerate software delivery, agile methodologies give priority to flexibility, collaboration, and iterative development. Here are how these principles lead to accelerated software development:

**Customer participation:** Agile techniques have customer involvement as one of their key aspects. This inclusion of customers in the team developing the software facilitates a more comprehensive understanding of what customers need or prefer. As such, the developed software matches closely the requirements that clients had to minimize extensive reviews or reworking in the final stages of project completion. Thus, feedback is incorporated incrementally resulting in faster cycles of development and quicker feature deployment.

**Incremental delivery:** Agile methods support incremental delivery of software functionalities through short time-boxed iterations known as “sprints”. Instead of waiting until they can deliver a complete system at once, agile teams provide working pieces periodically during a given period. This gradual release ensures that there is early-stage input from stakeholders and end-users which allows for assumption validation, gathering feedback, and adjustment as required before a conclusion on the development process has been reached. This means that it is possible to commence.

**People-oriented strategy:** In agile methods, the focus is put on individuals and their relations within a team or project instead of on processes and tools. Through fostering collaboration and adaptability in projects, Agile approaches empower cross-functional teams and encourage self-organization. It fosters open communication among team members who are encouraged to contribute toward decision-making by sharing ideas freely. The approach used in this direction ensures that the solution to the problem is found faster by enhancing decision-making procedures and efficient task accomplishment such as hastening the development pace.

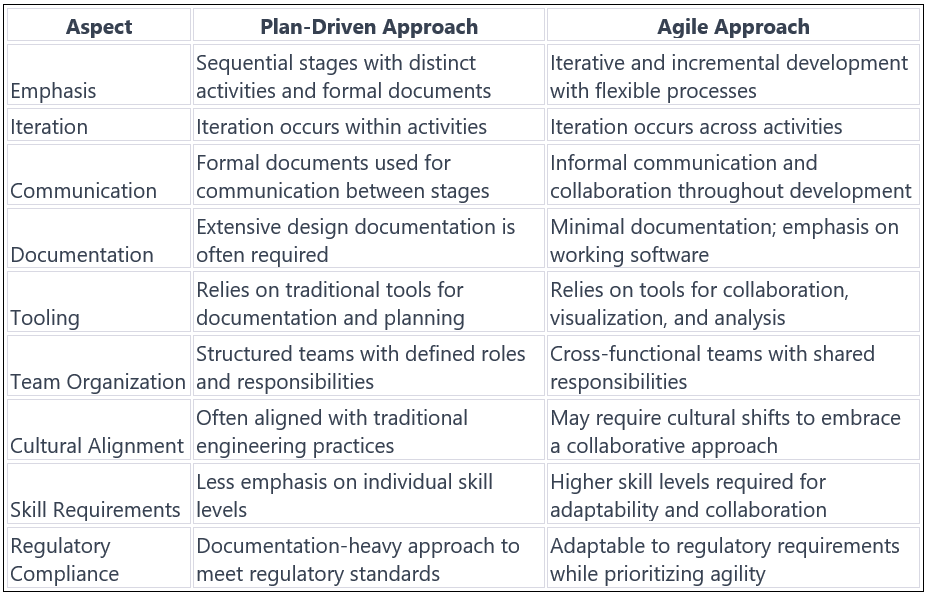
**Accepting change:** Agile methodologies understand that software development projects will always experience some changes. Agile teams consider this change an avenue for continuous improvement rather than opposing it. They are flexible enough to cope with changing requirements, priorities, or market conditions. The Agile process can indeed accommodate various needs existing in a given time frame or those which emerge from technological advancements thus ensuring maximum customer satisfaction through meeting ever-evolving demands effectively. Rapid adaptation to new insights, emerging technologies, and market shifts enables agile teams to respond quickly facilitating high-speed software delivery and deployment.

8.

A diagram of a process

Description automatically generated

The following are the key differences between the Plan-Driven Approach and the Agile Approach:



9.

When it is about scaling agile methods to larger projects developed by distributed development teams, there is also a need to consider adding some methods and documentation from plan-based approaches due to specific reasons:

**Increased Complexity:** Big projects usually range from multiple teams working on various parts of the system which in many cases may be in different areas and time zones. In such scenarios, the complexity of coordination and communication goes up with more planning and coordination needed.

**Interfacing with Existing Systems:** Large systems often referred to as “brownfield systems” do not exist in isolation but interact with numerous existing systems. The requirements for integration might not suit the flexible and incremental nature of agile development. Therefore, upfront design and planning are necessary to ensure that there is smooth integration with existing systems.

**Regulatory Compliance:** External regulations and standards frequently apply to large systems thus prescribing certain documents and procedures. Such regulatory needs may contradict agile methods that tend to have flexibility as well as informal communication. For this reason, including plan-based techniques helps guarantee the observance of regulatory standards.

**Cross-team Communication:** In distributed development environments, effective communication between teams becomes crucial. Plan-based approaches often emphasize formal documentation and communication channels, which can facilitate communication across distributed teams more effectively than informal agile practices.

**Levels of expertise and cultural disparities:** Big organizations may have different skill levels within their teams, and some may not be familiar with the Agile process at all. Introducing plan-based methods and documentation can provide a structured framework that is more familiar to individuals accustomed to traditional engineering processes. In addition, a culture that resists agile could result in a slow transition by incorporating some known plan-based practices.

**Change Management and Testing Standards:** Large organizations often have established change management and testing procedures that may conflict with agile principles such as refactoring and continuous testing. Adapting these processes to accommodate agile practices requires careful planning and integration, which may involve incorporating elements of plan-based approaches.

**References:**  
*Sommerville, Ian. Software Engineering. Pearson, 2011.*