**AI Section 7: Mapping Out the AI System and Software Development methods**

**If you want to follow along, please download [name of file]: [hyperlinked file]**

**Software Development Methods**

Objective

* Students will be able to identify the most common design methodologies for software development (agile, waterfall, and spiral) and some alternative options.
* Students will be able to identify and apply the 7 steps of program development cycle.
* Students will understand the distinction and importance of the types of documentation used in a program development cycle.
* Students will be able to identify the methods that collaborating developers use to manage different versions of a project.

Main Learning Goal

By the end of this lesson, students should be able to explain and apply the key components of pursuing a software development project; this includes common methodologies, major steps, documentation types, and version control.

Focus Question

How can we use software development processes to build projects?

A Long Time Ago

Suppose that you are a nomadic time traveler that decides to travel in the future to determine how differently the world operates compared to the current year, 1945. As you step out of the portal that led you to Ashgabat in the year 4922, you are astounded as you see that most of your surroundings are coated in chrome. The uniformity of the environment and the presence of cyborg-like beings are far beyond the wildest expectations that your fellow earthlings had in 1945. A local tour guide, Hajymyrat, approaches you and with certain reverence requests that you follow him. You tell the tour guide that you are an earthling from a long, long time ago.

As you wander through the futuristic streets of Ashgabat, you are told uncountable stories of the Software Development Struggles, which was a cataclysmic event that lasted from the year 2001 to 2023. You are perturbed to discover that earthlings in the year 2001 had failed to adhere to a consistent design methodology when working on software projects, thereby leading to innumerable projects that were left unfinished and ultimately, economic stagnation that resulted in widespread shortages of consumer goods. For example, historic records that were shown to you demonstrated that in the year 2021, a project centered on the now-antiquated concept of machine learning resulted in a catastrophe due to the development team’s rigid design methodology that left little time for gathering user input. These fatal choices led to significant amounts of irrelevant or poor data being fed into the natural language processing algorithm, resulting in a chatbot that while effective at imitating certain human responses, was never able to free itself of even trivial mistakes in arithmetic and responses to questions. In tears, you vow to avert such a catastrophe by warning your fellow earthlings about potential negligence in design methodologies. After having finished your tour, you decide to immediately return to your era to deliver the important message.

You are horrified to see that your time machine has failed to work and needs desperate repairs. The surrounding townsfolk fall under your command and form a software development team. Among your software development team members are seasoned programmers, managers and futuristic data scientists. The tour guide, however, reminds you of the mistakes of your descendants in 2001 and recommends that you determine the best possible design approach before beginning the repairs.

During the first meeting with your team, that is tasked with rebuilding the time machine to return to 1945, you are asked to answer the following questions:

1. Do we need active user input for this project? (Hint: Would building off team member’s ideas in an iterative process be beneficial?)
2. Is time of essence in this project? In other words, is this a project that is of a short-term or long-term nature?
3. Considering your answers to the previous questions and the design methodologies with which you are already familiar, which methodology would be the most logical to use in this scenario?

Software Development Processes

The term “software development process” is simply another way of referring to a software design methodology, which is an approach to developing software applications that forms an integral part of any software project. These methodologies allow teams to maintain their cohesion and maximize productivity by providing options for planning and developing a given software product. Common features of design methodologies include roadmaps, delegation of roles within teams and goal-based approaches. The project at hand often determines which design methodology works best, as projects will often not have the exact same needs and teams.

We will cover a few examples of the most common design methodologies below. Do keep in mind that this coverage is not exhaustive. We will cover other types of design methodologies and provide a deeper insight into the software development process in a later module.

Agile Development

Perhaps the most well-known design methodology today, the agile design methodology is highly flexible and an iterative approach that emphasizes user feedback at every step of the process. This design methodology is most suited for projects that need constant feedback from stakeholders and are expected to change in terms of requirements. As such, using this design methodology means that the development team must be ready for unpredictability and adapt to the needs of the stakeholders as they evolve during the course of development. To achieve this task, the agile development methodology stipulates that the project should be divided into small development steps called iterations, with each iteration involving a round of user feedback.

For example, suppose that you are in a team that wants to develop some desktop application. Using the agile methodology, the team may begin by focusing on a narrow range of features, such as the user interface and authentication. Once this set of features has been implemented, the team would acquire feedback from the client and begin the next iteration of the development cycle, where feedback from the client is taken into account and necessary changes are made before moving on to the next set of features to develop. The application would continue to evolve and iterations rapidly performed until the customer’s feedback indicates that the application meets all of the requirements.

Waterfall Methodology

Another popular design methodology is waterfall, which in many cases is considered to be the opposite of the agile methodology. Rather than emphasizing rapid iterations that focus on unpredictability and customer feedback, the waterfall methodology focuses on rigid planning and documentation. As such, we can consider this methodology to be a linear approach to software development; the team obtains all of the customer’s requirements before development begins and delivers a product that is based on these defined requirements. The methodology is not adaptable to sudden changes, as the team’s primary goal is to adhere to its initial goals. Documentation that covers the software’s functionality is a key component that is incorporated into the development cycle. In this approach, the team’s development cycle is divided into stages that do not overlap and consist of designing, implementing, testing, deploying and maintaining the given software product. Consequently, we can see that this methodology’s primary disadvantage is its inflexibility, as any changes that must be made could be costly to the project and require a completely new start.

An example of this methodology’s applicability is with a system that requires strict protocols due to the need to adhere to safety standards. For example, designing software for a life support system in spacecraft would require extensive documentation and testing throughout the development cycle. Furthermore, requirements would need to be extremely detailed and cover all necessary aspects of the

Spiral Model

The Spiral model combines elements of both the waterfall and agile design methodologies. It is particularly useful for large and complex projects with high levels of uncertainty or risk that at the same time require extensive planning and user feedback. In this approach, the development process is divided into cycles, with each cycle or “spiral” having its own planning, design, implementation and testing phases. These cycles allow for a more gradual and controlled approach to development, with risk assessment and mitigation being the primary focus within each cycle.

We can continue from the previous example for the waterfall methodology to describe an application of the spiral model. Rather than focusing on a life support system, suppose that we are building software for human habitation on Mars, which would include spaceflight. We can assume that due to the nature of the mission, there would be significant risk and constantly changing requirements as new challenges are discovered. In this situation, implementing an exclusively agile or waterfall approach would be undesirable as neither methodology can by itself compensate for its inherent disadvantages.  Instead, we could use the spiral model to tackle each challenge that can arise during the development of these systems and test each component while building these components incrementally and assessing any possible risks at every step.

Alternative Methods of Program Development

While we have already covered the most important and well-known methods of program development, there are several other alternatives that may be worth considering under certain circumstances that are unique to the given project.

Peer Coding

Peer coding, also known as pair coding or peer programming, is an approach to program development that emphasizes collaboration between two programmers while building a product. These two programmers will often work in the same environment and have specialized roles, which are the “driver” and the “observer.” The “driver” of this approach will be the individual that implements the code for the given software system, while the “observer” will continuously review the code and implementation while providing active feedback.

An example of a possible use of peer coding is when writing extensive algorithms for a given task. Suppose that a team of two programmers decides that the best approach towards developing a website for a hotel booking agency would be to carry out peer coding. One of the programmers would then write the code for the website while the other programmer would oversee all progress and root out any errors in logic or implementation. This approach would therefore ensure that both members of the team are focused on the latest changes to the project and are thereby fully knowledgeable regarding its state.

Rapid Prototyping

As implied by its name, rapid prototyping is an approach that involves the rapid creation of an early version or protype of a given software system. The prototype would then be used as an example of the ideal structure of the software system such that it becomes possible to gather user feedback and improve the prototype’s design. Rapid prototyping is often used when the requirements of the product are not particularly clear and need to be further clarified by providing an initial, barebones version of what the development team envisions.

For example, suppose that your team has been tasked with developing a social media platform, but the requirements of the final product are not particularly well defined. Rather than devoting significant amounts of time and resources into the development of the full product, the team may use rapid prototyping to create a “barebones” version of the social media platform that only contains basic functionalities such as making posts and comments. With this prototype, the team can then gather feedback from the customer and other potential stakeholders to determine what features need to be removed or implemented.

Extreme Programming (XP)

Extreme Programming is considered to be a type of agile methodology that has an even greater emphasis on short development cycles and little documentation at every cycle with the goal of implementing a quality software product against rapidly changing requirements. There are four key concepts in extreme programming: coding, testing, listening and designing. Given this context, this approach can be seen as one that devotes great focus on teamwork and communication, with integration of software components, testing and deployment occurring at a fast pace.

An example of a use case for extreme programming would be the development of a website that has diverse functionalities. The development team would then opt to rapidly write code for one functionality and integrate it within the website. As each of these components are implemented, test cases would be written and used. This constant cycle of implementation and immediate testing, known as continuous integration, forms a key part of extreme programming. The team would ultimately be able to rapidly identify issues before other components are considered.

Feature-Driven Development (FDD)

Feature-driven development is a software development process that focuses on breaking down a software system into discrete, well-defined features; that is, FDD follows a structured approach where development begins by identifying, designing and implementing individual features of the software. Each feature is thoroughly analyzed, with a detailed design being created for each feature. The development team would then begin implementing each of these features.

The feature-driven development methodology is divided into the following five stages:

* Model or Prototype Development -> In this stage, the team creates a high-level design of the system’s structure.
* Feature Identification -> Each feature of the system is described and considered. The result of this stage is a list of features that are to be developed.
* Plan by Feature -> For each feature, the team devises a plan that outlines the design and tasks that are needed to implement the features.
* Design by Feature -> Detailed designs are created for each feature. Designers and developers work collaboratively to ensure that the feature is well-designed.
* Build by Feature -> In this final stage, the team develops and integrates each feature into the system. This process is done per feature to provide ample room for test cases.

A classic example of a use case for FDD would be a team seeking to develop a shopping website. Before building the website, the team would first identify and list important features that are relevant to the project, such as the payment system, user accounts, the shopping cart and a search feature. With each feature now being considered, the team would take one feature at a time and develop, test and deploy it. This approach can help teams break down massive projects into simpler tasks that provide many opportunities to effectively plan and test each feature before its eventual integration into the system.

Steps in the Program Development Cycle

The program development cycle, also known as the software development lifecycle (SDLC), is a systematic process that provides a guide to the creation of a software system from the beginning stages to the deployment and maintenance of the product. The SDLC often consists of six important steps that are as follows:

1. **Analysis** -> In this first stage, the development team and potential stakeholders begin the planning process by identifying and considering the problem at hand and how a software product could address the given problem. The analysis stage primarily involves determining user requirements, establishing goals, delegating roles and providing a roadmap for the development of the product. This stage is indispensable, for the base and direction of the project are established here.
2. **Design** -> The design phase takes the initial ideas gathered during the analysis stage and expands upon them by spurring the creation of a vision for the software product’s structure. This structure includes features such as data structures to be used, user navigation and other components of the system that are deemed key for the success of the product’s development. In essence, the team focuses on planning for each part of the system and how it is to be developed in accordance with the requirements of the project.
3. **Implementation** -> After the design has been completed, the team begins to write the code for the software system in accordance with the requirements outlined in previous steps. More specifically, this step involves integrating system components and implementing the functionality of the software.
4. **Testing** -> Creating test cases for software components is an extremely important task in any project, for it provides teams with the chance to evaluate the software that they developed. This step therefore involves the identification and elimination of any possible errors or shortcomings in logic that may have occurred during the previous step. This step also involves various types of testing that will be covered in greater detail in a later module, such as unit testing and user acceptance testing.
5. **Deployment** -> After the system is thoroughly tested and is deemed deployable, the team moves on to this step. In essence, “deploying” a software product means making the product available for users. Deployment can be performed through various means that will depend on the nature and goals of the project, such as deployment to a local machine or to a mobile device.
6. **Maintenance** -> While the deployment step signals that the product is finished, it does not mean that the team simply moves on to developing another product. After deployment, the software needs to be continuously supported through updates that fix bugs and other issues based on direct user feedback. This step in itself can be considered to be a cycle, as software products can continue to be supported for many years if they are a commercial success.

Documentation Types Used in the Program Development Cycle

Documentation provides a clear and organized way of recording the project’s needs, general design, test cases and other information that may be useful for anyone who uses the product in the future. As such, documentation is often considered to be a core component of the program development cycle, owing to its importance as a “record-keeping” feature that keeps track of the development team’s interaction with the developing product.  Some of the most important documentation types used in the program development cycle are described below. Before continuing, it is important to know that all software documentation types are divided into two main categories: product documentation and process documentation.

Product Documentation

The product documentation is structured as a general guide that contains instructions for how to interact with the functionalities of the system. In essence, we are describing the product that is being developed.

Product documentation can be divided into the following components:

***System Documentation***

System documentation provides descriptions of the system and its corresponding components. Some of the documents that would fall under system documentation are as follows:

* Requirements Documents -> These documents outline the general requirements of the project that are defined at the beginning of the development cycle.
* Design Decisions -> This document provides an insight into the design choices that were made by the team, which details the product’s structure.
* Architecture Descriptions -> These provide an overview of the system's architecture, including its components.
* Program Source Code -> The source code explains the code’s functionality and logic, which can be a useful reference for future developers.

***User Documentation***

User documentation provides information for users of the given product, which can range from instructions to more detailed information that can help users navigate the product. Some of the documents that would fall under this category include the following:

* Tutorials -> These help users understand how to use the product.
* User Guides -> These can provide more detailed information regarding the product’s functionalities.
* Troubleshooting Manuals -> These can provide solutions to common problems that may be encountered while using the product.
* Installation Manuals -> These commonly include step-by-step instructions for installing the product.

Process Documentation

Process documentation details all of the processes that describe the development and maintenance cycle of the given system, such as meeting notes, project plans and test cases used.  Common examples of process documentation include the following:

* Project Plans -> These outline the scope, objectives and timeline of the development project, thereby serving as a roadmap.
* Test Schedules -> These provide a detailed timeline of testing procedures.
* Reports -> Document the findings and progress of the development process, which offers a record of the project's current state.
* Standards -> The standards outline the coding structure rules to which developers must adhere.
* Meeting Notes -> These include all discussions and decisions made by the team.
* Correspondence -> Includes communication related to the project, whether in the form of electronic (e.g., email) correspondence or other means of communication.

Version Control Methods

Version control is an indispensable task within software development that you will encounter in all areas of the professional world. Version control tools allow you to collaborate with other team members on a project while tracking all changes and maintaining different versions of the software in the case that a rollback is needed. Some of the most common tools an essential aspect of software development that enables multiple developers to collaborate on a project, track changes, and manage different versions of the software. Various version control methods and systems are used in the industry to facilitate this process. Let's explore some of the primary version control methods:

Git

Perhaps the most famous and commonly used version control tool, Git is a tool that allows developers to work on a shared repository that keeps records of past versions of the code. Using Git allows you to work in unison with other developers and thereby facilitates collaboration.

Subversion (SVN)

Subversion, also known as SVN, is a centralized version control system, which means that it maintains a central repository that stores the entire project's history, making it suitable for projects that seek to have as much rollback capability as possible while maintaining control from a central location. SVN tracks all changes that are made to files and directories, which makes it possible to revert back to older versions of the codebase, much like with Git and other tools.

Mercurial

Mercurial is another distributed version control system similar to Git, which means that it still offers similar flexibility that is almost always needed for collaboration in software development. Much like other tools, Mercurial allows you to work independently on a local repository and merge changes when they are ready.

Case Study: To Infinity and Beyond

Click the link to go to the assignment.