The Software Development Life Cycle (SDLC) and Related Methodologies:

An Analysis of SDLC, Iterative and Prototyping Methods, and Modularization in SDLC Design

Dan Bailey

SDEV120 Computing Logic

Michael Heady

September 4, 2025

Abstract

This paper examines the Software Development Life Cycle (SDLC) as a structured process for creating quality software. It outlines the seven main stages of the SDLC, from understanding the problem to executing and maintaining the final product. This paper explores alternative methods, specifically iterative and prototyping, comparing their methods to building and refining software. Modularization is also analyzed, explaining how breaking down programs into smaller, manageable components helps maintainability and scalability. The paper points out that these structured processes and design principles are crucial to successful software engineering.

The Software Development Life Cycle and Methodologies: An Analysis of SDLC, Iterative and Prototyping Methods, and Modularization in SDLC Design

Creating a computer program, from a simple mobile app to a complex ordering system, requires a well-thought-out plan. Just like building a house needs a blueprint, building software needs a structured process to avoid mistakes and to make sure the final product works correctly. This process is known as the Software Development Life Cycle (SDLC). This paper will explain the steps of the SDLC, look at different ways to use it, like iterative and prototyping methods, and discuss the importance of organizing code into smaller pieces, a practice called modularization.

The Software Development Life Cycle (SDLC)

The Software Development Life Cycle is a series of steps that programmers follow to create high-quality software. The main goal of the SDLC is to provide a clear, step-by-step process that helps ensure the final program is well-planned, bug-free, and does what the user wants it to do. Following these steps will help keep a project on time and on budget. According to Farrell (2023), the process involves seven main stages:

- **Understanding the Problem:** This is the first and most critical step. Developers talk to the users to figure out exactly what the software needs to accomplish.
- **Planning the Logic:** Before writing any code, the programmer plans the solution. This involves creating flowcharts or pseudocode to outline the program's steps and logic.
- **Coding the Program:** The programmer writes the code in a specific programming language, turning the planned logic into instructions the computer can understand.
- **Translating the Program:** The written code is converted into machine language that the computer can execute. This is usually done by a compiler or interpreter.
- **Testing the Program:** The software is tested thoroughly to find and fix any errors or "bugs." This makes sure the program works as it should and produces correct results.
- **Putting the Program into Production:** Once the software is tested and working correctly, it is deployed for people to use.
- **Maintain the Program:** This step involves making updates or fixing any new problems that arise. The goal of the maintenance phase is to ensure the software remains functional, secure, and useful for as long as it is in production.

The SDLC is important because it brings structure to the complex task of software development, reducing the chances of failure and leading to a more reliable product.

Iterative & Prototyping Methods

While the SDLC provides a map, there are different ways to travel it. Two common methods are iterative and prototyping methods.

The iterative method is a tactic where a program is built in small sections, or "iterations." Instead of building the entire system at once, developers create a basic working version and then add more features in repeating cycles. Each cycle adds a new piece of functionality, which is then tested and refined. This method is flexible and allows for customer feedback throughout the process.

Prototyping involves creating a sample or a mockup of the program to show to users before building the full program. A prototype isn't meant to be a final product. Its purpose is to get feedback on the design and usability. For example, when designing a fast-food restaurant ordering system, a team might make a simple prototype of the kiosk screen. Customers could try it out and provide feedback on whether the buttons are easy to understand and if the ordering process is smooth. This helps programmers identify design flaws early, saving time and money.

The main difference between the two is that the iterative method produces a working part of the program with each cycle, while prototyping creates a model that is usually thrown away once it has served its purpose of gathering feedback (GeeksforGeeks, 2023).

Modularization & Program Design

Good program design involves breaking a large problem down into smaller, manageable pieces. This is called modularization. A module is a self-contained part of a program that performs a specific task. For example, when developing an online event registration form, the program can be split into several modules:

- A module to handle user input
- A module to validate that the information is correct.
- A module to process the payment.
- A module to send a confirmation email.

Modularization is important because it makes code easier to manage. If there is a bug in the payment processing, the programmer only needs to examine the payment module instead of the entire program. This makes debugging and maintenance a lot easier. It also makes the program more scalable. If the registration form needs a new feature, like a share button, a new module can be added without having to rewrite the existing code. Good modules are designed to be reusable, so the "send confirmation email" module could be used in other programs as well.

Conclusion

The Software Development Life Cycle provides the crucial structure needed to guide a programming project from an idea to a finished product. By following its stages, programmers can make sure they create logical and effective solutions. Modern approaches like prototyping allow for early user feedback, while the core design principle of modularization helps keep code organized, easy to maintain, and update. These practices are essential to creating successful and reliable software.

References

Farrell, J. (2023). *Programming logic and design* (10th ed.). Cengage Learning.

GeeksforGeeks. (2023, October 12). *Software engineering | Prototyping model*. GeeksforGeeks. https://www.geeksforgeeks.org/software-engineering-prototyping-model/