What is Software Development Life Cycle

The Software Development Life Cycle is a blanket term for methods of planning and executing software development while attempting to mitigate cost and length of production. Usually, large scale development processes require the organization and input of many different agents. Clients, engineers, developers and corporate executives could all be involved in the process of designing, funding and creating a development project. Proceeding with a well structured plan can mitigate the confusion, cost and deadline pitfalls that would occur without such a structure.

The Software Development Life Cycle was developed formerly in the 1960s to the 1970s[1] in tandem with Structured Programming and Object-Oriented Programming. As large scale businesses were making use of computers to process large amounts of data, it was concluded that systems would have to be created to standardize software development. Much like any other engineering field, a standardized environment with a methodical approach would mitigate future confusion through the inevitable development of more eclectic programming styles.

Today we have many different methodologies, however all of them are rooted in the original Waterfall Model created by Winston W. Royce in 1970. Though widely criticized, it was based on traditional engineering Systems Development Life Cycles and laid the groundwork for future models.

Phases of a Typical Software Development Life Cycle

Requirements and Analysis

This is the planning phase. During this time, the clients and companies determine what they need from their developed software. They they determine what would be cost prohibitive and the desired time frame. Initial relationships are built between clients, executives and developers.

Design

During this phase the design of the software solution is determined. This is where developers sit down with clients and hash out what the clients need and establish how these needs can be met within their previously established limits.

Implementation and Coding

The development team gathers to put forth a plan for execution. A time frame is established with various goals along milestones to the project's completion. Team members are given tasks usually by a Development Lead and regular updates are given to ensure the project is being developed within the established time frame.

Testing

Once the initial coding is finished quality assurance teams test the ‘qual’ product thoroughly for any code defects, bugs or negative user experiences. If further development is needed, the code is fixed or rewritten and goes back to quality assurance for testing.

Deployment

Once the software is finished with quality assurance and it is deemed finished with the client’s specifications in mind, it is considered ready for production. The software is usually ‘rolled’ into production and the client is notified that their software is now live.

Maintenance

Continued maintenance is required after deployment. As thorough as quality assurance may be, end users will inevitably find bugs or other issues with the software that will need to be addressed. When this occurs, the SDLC starts again with regard to the solution for a fix, culminating in a new production release.

Software should be written with this final phase in mind. If the code is written in a functional manner with comments and a simple, easy to understand execution, maintenance will be much easier. This, in turn keeps cost of production lower and gives development time to develop client feature requests.

Even if software has been executed with every factor in mind and tested exhaustively, the changes in the way we interface with and use software at large will require maintenance. This phase is often given too little attention and could eventually result in what is called ‘tech debit’ down the road. Enough tech debit could eventually result in a very costly software rewrite and overhaul that could have been prevented with better foresight of the Maintenance phase.

Methodologies

Waterfall

The Waterfall Methodology is the original and simplest methodology. It quite simply takes the above steps and follows them in sequential order to completion. Typically clients and executives are only consulted once at the beginning of the design phase. This methodology has been criticized for this as the development will inevitably go over budget and not reach their deadline. If there is no adjustment for these two factors during or after the coding and testing process, the client is left in the dark until the development is completed.

Also, if the client's needs change prior to deployment, the product will need to be changed after the initial code has already been established. This could result in an enormous undertaking if the client's change dictates a dramatic shift from the initial design or requirements.

Agile

The Agile methodology mimics the Waterfall model, however it does so with continuous customer interaction. This interaction coincides with continuous small releases working in an iterative fashion until a final finished product is deployed. This method is very popular for its transparency and client interaction. It can however cause the development to go in an undesired direction and especially in a Scrum environment can be taxing for the developers.

With continuous releases the design process can be bogged down with many feature requests before a working model is created. Given that it tends to lend itself to faster paced development, shortcuts in code are often used that create complex code that becomes difficult to maintain.

Clients also prefer an Agile methodology as they have direct control over the development team's timeline and costs during coding and testing. The increased executive and client interaction has its benefits and disadvantages, but is generally seen as a preferred way of developing more complex software platforms.

Spiral

The Spiral methodology simplifies the process into four phases: planning, risk analysis, coding and evaluation. These four phases repeat until all parties are satisfied with the result. Only then is it deployed. The primary advantage to the Spiral methodology is risk management. Each phase examines the current status of the development and rewrites its direction with risk in mind. This is a methodology more often used in very large corporate environments that must take many different individuals' feedback but do so in a way that allows developers to proceed with a defined goal.

When large-scale budgeting and resource management is something that must be considered, the Spiral methodology is preferred for its flexible rigor. While not as flexible as an Agile approach, the Spiral methodology allows flexibility in it's iterative cycle.

Conclusion

There are many more methodologies than the three listed here, but these are the most common. Ultimately, every software development takes its own approach, often resulting in a unique hybrid that tailors itself to the needs of all the individuals involved. The most important reason for having a SDLC methodology is establishing a mindful approach to development to maximize efficiency and reduce present and future development costs.

Cited Sources

[1]Wikipedia. Software development process, 2017, https://en.wikipedia.org/wiki/Software_development_process. Accessed 5 Dec. 2017.

[2]IT Knowledge portal. Software Development Methodologies, 2017, http://www.itinfo.am/eng/software-development-methodologies/. Accessed 5 Dec. 2017.

[3]TutorialsPoint. Software Development Life Cycle, 2017, https://www.tutorialspoint.com/software_engineering/software_development_life_cycle.htm. Accessed 5 Dec. 2017.