stackify.com /what-is-sdlc/

What is SDLC? Understand the Software Development Life Cycle

6-8 minutes

Streamlined development relies on a consistent methodology and a clearly-defined process from getting from point A to point B. If you're just getting your feet wet in the wide world of development, you need to understand the Software Development Life Cycle or SDLC.

Definition of SDLC

SDLC or the Software Development Life Cycle is a process that produces software with the highest quality and lowest cost in the shortest time. SDLC includes a detailed plan for how to develop, alter, maintain, and replace a software system.

SDLC involves several distinct stages, including planning, design, building, testing, and deployment. Popular SDLC models include the waterfall model, spiral model, and Agile model.

How SDLC Works

SDLC works by lowering the cost of software development while simultaneously improving quality and shortening production time. SDLC achieves these apparently divergent goals by following a plan that removes the typical pitfalls to software development projects. That plan starts by evaluating existing systems for deficiencies. Next, it defines the requirements of the new system. It then creates the software through the stages of design, development, testing, and deployment. By anticipating costly mistakes like failing to ask the end user for suggestions, SLDC can eliminate redundant rework and after-the-fact fixes.

Stages and Best Practices of SDLC

Following the best practices and/or stages of SDLC ensures the process works in a smooth, efficient, and productive way.

1. **Identify the** current problems**.** "What don't we want?" This stage of SDLC means getting input from all stakeholders, including customers, salespeople, industry experts, and programmers. Learn the strengths and weaknesses of the current system with improvement as the goal.
2. **Plan.** "What do we want?" In this stage of SDLC, the team defines the requirements of the new software and determines the cost and resources required. It also details the risks involved and provides sub-plans for softening those risks. In this stage, a Software Requirement Specification document is created.
3. **Design.** "How will we get what we want?" This phase of SDLC starts by turning the software specifications into a design plan called the Design Specification. All stakeholders then review this plan and offer feedback and suggestions. It's crucial to have a plan for collecting and incorporating stakeholder input into this document. Failure at this stage will almost certainly result in cost overruns at best and total collapse of the project at worst.
4. **Build.** "Let's create what we want." This SDLC stage develops the software by generating all the actual code. If the previous steps have been followed with attention to detail, this is actually the least complicated step.
5. **Test.** "Did we get what we want?" In this stage, we test for defects and deficiencies. We fix those issues until the product meets the original specifications.
6. **Deploy.** "Let's start using what we got." Often, this part of the SDLC process happens in a limited way at first. Depending on feedback from end users, more adjustments can be made.
7. **Maintain.** "Let's get this closer to what we want." The plan almost never turns out perfect when it meets reality. Further, as conditions in the real world change, we need to update and advance the software to match.

The DevOps movement has changed the SDLC in some ways. Developers are now responsible for more and more steps of the entire development process. We also see the value of shifting left. When development and Ops teams use the same toolset to track performance and pin down defects from inception to the retirement of an application, this provides a common language and faster handoffs between teams. APM tools can be used in development, QA, and production. This keeps everyone using the same toolset across the entire development lifecycle.

Read More: 3 Reasons Why APM Usage is Shifting Left to Development & QA

Examples of SDLC in Action

The most common SDLC examples or SDLC models are listed below.

- **Waterfall Model.** This SDLC model is the oldest and most straightforward. With this methodology, we finish one phase and then start the next. Each phase has its own mini-plan and each phase "waterfalls" into the next. The biggest drawback of this model is that small details left incomplete can hold up the entire process.
- **Agile Model.** The Agile SDLC model separates the product into cycles and delivers a working product very quickly. This methodology produces a succession of releases. Testing of each release feeds back info that's incorporated into the next version. According to Robert Half, the drawback of this model is that the heavy emphasis on customer interaction can lead the project in the wrong direction in some cases.
- **Iterative Model.** This SDLC model emphasizes repetition. Developers create a version very quickly and for relatively little cost, then test and improve it through rapid and successive versions. One big disadvantage here is that it can eat up resources fast if left unchecked.
- **V-Shaped Model.** An extension of the waterfall model, this SDLC methodology tests at each stage of development. As with waterfall, this process can run into roadblocks.
- **Big Bang Model.** This high-risk SDLC model throws most of its resources at development and works best for small projects. It lacks the thorough requirements definition stage of the other methods.
- **Spiral Model.** The most flexible of the SDLC models, the spiral model is similar to the iterative model in its emphasis on repetition. The spiral model goes through the planning, design, build and test phases over and over, with gradual improvements at each pass.

Benefits of SDLC

SDLC done right can allow the highest level of management control and documentation. Developers understand what they should build and why. All parties agree on the goal up front and see a clear plan for arriving at that goal. Everyone understands the costs and resources required.

Several pitfalls can turn an SDLC implementation into more of a roadblock to development than a tool that helps us. Failure to take into account the needs of customers and all users and stakeholders can result in a poor understanding of the system requirements at the outset. The benefits of SDLC only exist if the plan is followed faithfully.