waterfall model

Waterfall Model: What Is It and When Should You Use It?

December 8, 2016 | Andrew Powell-Morse | SDLC

First introduced by Dr. Winston W. Royce in a paper published in 1970, the `waterfall model` is a software development process. The waterfall model emphasizes that a logical progression of steps be taken throughout the software development life cycle (SDLC), much like the cascading steps down an incremental waterfall. While the popularity of the waterfall model has waned over recent years in favor of more `agile` methodologies, the logical nature of the sequential process used in the waterfall method cannot be denied, and it remains a common design process in the industry.

Throughout this article we'll examine what specific stages make up the core of the waterfall model, when and where it is best implemented, and scenarios where it might be avoided in favor of other design philosophies.

Some more specific takes on SDLC include:

Rapid Application Development	Test-Driven Development	Software Development Life Cycle
Iterative Model	Extreme Programming	Scaled Agile Framework
Agile Model	Scrum	Rational Unified Process
Big Bang Model	V-Model	Conceptual Model
Kaizen Model	Kanban Model	Spiral Model

The Six Stages of Falling Water

Actually implementing a waterfall model within a new software project is a rather straightforward process, thanks in large part due to the step-by-step nature of the method itself. There are minor differences in the numbers and descriptions of the steps involved in a waterfall method, depending on the developer you ask (and even the year during which you ask him or her). Regardless, the concepts are all the same and encompass the broad scope of what it takes to start with an idea and develop a full-scale, live application.

- **Requirements**: During this initial phase, the potential requirements of the application are methodically analyzed and written down in a specification document that serves as the basis for all future development. The result is typically a `requirements document` that defines *what* the application should do, but not *how* it should do it.
- **Analysis**: During this second stage, the system is analyzed in order to properly generate the models and business logic that will be used in the application.
- **Design**: This stage largely covers technical design requirements, such as programming language, data layers, services, etc. A design specification will typically be created that outlines how exactly the business logic covered in `analysis` will be technically implemented.
- **Coding**: The actual source code is finally written in this fourth stage, implementing all models, business logic, and service integrations that were specified in the prior stages.
- **Testing**: During this stage, `QA`, beta testers, and all other testers systematically discover and report issues within the application that need to be resolved. It is not uncommon for this phase to cause a "necessary repeat" of the previous `coding` phase, in order for revealed bugs to be properly squashed.
- **Operations**: Finally, the application is ready for deployment to a live environment. The `operations` stage entails not just the deployment of the application, but also subsequent support and maintenance that may be required to keep it functional and up-to-date.

The Advantages of the Waterfall Model

While the waterfall model has seen a slow phasing out in recent years in favor of more `agile` methods, it can still provide a number of benefits, particularly for larger projects and organizations that require the stringent stages and deadlines available within these cool, cascading waters.

- **Adapts to Shifting Teams**: While not necessarily specific to the waterfall model only, using a waterfall method does allow the project as a whole to maintain a more detailed, robust scope and design structure due to all the upfront planning and documentation stages. This is particularly well suited to large teams that may see members come and go throughout the life cycle of the project, allowing the burden of design to be placed on the core documentation and less on any individual team member.
- **Forces Structured Organization**: While some may argue this is a burden rather than a benefit, the fact remains that the waterfall model *forces* the project, and even the organization building said project, to be extraordinarily disciplined in its design and structure. Most sizable projects will, by necessity, include detailed procedures to manage every aspect of the project, from design and development to testing and implementation.
- **Allows for Early Design Changes**: While it can be difficult to make design changes later in the process, the waterfall approach lends itself well to alterations early in the life cycle. This is great when fleshing out the specification documents in the first couple stages with the development team and clients, as alterations can be made immediately and with minimal effort, since no coding or implementation has actually taken place up to that point.
- **Suited for Milestone-Focused Development**: Due to the inherent linear structure of a waterfall project, such applications are always well-suited for organizations or teams that work well under a milestone- and date-focused paradigm. With clear, concrete, and well understood stages that everyone on the team can understand and prepare for, it is relatively simple to develop a time line for the entire process and assign particular markers and milestones for each stage and even completion. This isn't to suggest software development isn't often rife with delays (since it is), but waterfall is befitting the kind of project that needs deadlines.

The Disadvantages of the Waterfall Model

While some things in software development never really change, many others often fall by the wayside. While Dr. Royce's initial proposal of what is now known as the waterfall model was groundbreaking when first published back in 1970, over four decades later, a number of cracks are showing in the armor of this once heralded model.

- **Nonadaptive Design Constraints**: While arguably a whole book could be written on this topic alone, the most damning aspect of the waterfall model is its inherent lack of adaptability across all stages of the development life cycle. When a test in stage five reveals a fundamental flaw in the design of the system, it not only requires a dramatic leap backward in stages of the process, but in some cases, can be often lead to a devastating realization regarding the legitimacy of the entire system. While most experienced teams and developers would (rightfully) argue that such revelations shouldn't occur if the system was properly designed in the first place, not every possibility can be accounted for, especially when stages are so often delayed until the end of the process.
- **Ignores Mid-Process User/Client Feedback**: Due to the strict step-by-step process that the waterfall model enforces, another particularly difficult issue to get around is that user or client feedback that is provided late into the development cycle can often be too little, too late. While project managers can obviously enforce a process to step back to a previous stage due to an unforeseen requirement or change coming from a client, it will be both costly and time-consuming, for both the development team and the client.
- **Delayed Testing Period**: While most of the more modern `SDLC` models attempt to integrate testing as a fundamental and always-present process throughout development, the waterfall model largely shies away from testing until quite late into the life cycle. This not only means that most bugs or even design issues won't be discovered until very late into the process, but it also encourages lackadaisical coding practices since testing is only an afterthought.

In spite of going through an explicit testing phase during implementation of a `waterfall model` project, as discussed above, this testing is often too little, too late. *In addition to* the normal testing phase, you and your team should strongly consider introducing an effective error management tool into the development life cycle of your project. Airbrake's error monitoring software provides real-time error monitoring and automatic exception reporting for all your development projects. Airbrake's state of the art web dashboard ensures you receive round-the-clock status updates on your application's health and error rates. No matter what you're working on, Airbrake easily integrates with all the most popular languages and frameworks. Plus, Airbrake makes it easy to customize exception parameters, while giving you complete control of the active error filter system, so you only gather the errors that matter most.

Check out Airbrake's error monitoring software today and see for yourself why so many of the world's best engineering teams use Airbrake to revolutionize their exception handling practices!

Related

Iterative Model: What Is It And When Should You Use It?
December 15, 2016
In "SDLC"

V-Model: What Is It And How Do You Use It?
December 26, 2016
In "SDLC"

Rapid Application Development (RAD): What Is It And How Do You Use It?
November 23, 2016
In "SDLC"