Software Processes

V-Model

Adaptive Software Development

Software Engineering 2018

Bruno Carvalho
up201606517

Diogo Yaguas
up201606165

Tiago Castro
up201606186

October 25, 2018

Contents

Adaptive Software Development

Adaptive software development (ASD) is an iterative, circular process of software development proposed by Jim Highsmith and Sam Bayer as a technique for building complex software and systems published in 2000. It's focused on rapid creation and evolution of software systems and is a predecessor of modern agile development techniques.

The Process of Adaptive Software Development

Adaptive replaces old waterfall-like models with a repeating series of *speculate*, *collaborate* and *learn* cycles. It is characterized by constant change, re-evaluation, peering into an uncertain future and intense collaboration among developers, testers and customers.

Speculation

This is the setup phase. During it, developers attempt to understand the exact nature of the product and requirements of the users and customer. Because the process is inherently circular, a previous cycle feeds this phase with new information such as bugs, user reports and new requirements found during development by the team and the customer.

Speculation comprises *project initiation* and *risk-driven cycle planning*.

Project Initiation The project's constraints and *mission statement* are set by the customer, while the team determines the project's organization, gathers more requirements, and estimates size and scope.

The *mission statement* needs to be specific, yet flexible and focused. Attempting to excel in multiple dimensions usually results in a product being mediocre in all of them and excellent in none.

Risk-Driven Cycle Planning The mission statement's aim is to expose the customer's desired result for the project. Most likely the customer has no knowledge of the field, so this can be done adequately in layman's terms. Using the mission statement, the team forces it's focus on a single area — such as features, schedule, defects, or resources — in which it must excel in the development phase.

Finally, the team fixes a set of features to implement in the next iteration and also *fixes* a certain amount of time for it.

Collaboration

This is the development phase. Parallel efforts include (minor) adjustments due to changing technologies, requirements, stakeholders

V-Model

The V-Model is a unique, linear development methodology, defined by the late Paul Rook in the 1980s, used during a software development life cycle. It was accepted in Europe and UK as a progressive alternative to the Waterfall model.

The V-Model focuses on fairly typical waterfall-esque method that follows strict, step-by-step stages. While initial stages are broad design stages, progress proceeds down through more and more granular stages, leading into implementation and coding, and finally back through all testing stages prior to completion of the project.

The Process of V-Model

Much like the traditional waterfall model, the V-Model specifies a series of linear stages that should occur across the life cycle, one at a time, until the project is complete. For this reason, V-Model is not considered an agile development method, and due to the sheer volume of stages and their integration into the overall process, understanding the model in detail can be challenging for everyone on the team, let alone the client or users.

The V-shape of the V-Model represents the various stages that will be passed through during the software development lifecycle. Beginning at the top-left stage and working, over time, toward the top-right tip, the stages represent a linear progression of development rather similar to the waterfall model.

So V-Model contains Verification phases on the left and Validation phases on the right of the development phase — hence the name.

Design Phase

Requirements Gathering and Analysis Contains detailed communication with the customer to understand the requirements and their expectations.

System Design Contains the system design and the complete hardware and communication setup for developing the product.

Architectural Design System design is broken down further into modules taking up different functionality. The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood.

Module or Low-Level Design Here the detailed design of each module is specified.

Testing Phases

Unit Testing Unit Test Plans are developed during the module design phase. These Unit Test Plans are executed to eliminate simple bugs at code or unit level.

Integration testing After completion of unit testing come integration tests, which verify the communication of modules among themselves.

System Testing Tests the application as whole, including functionality, interdependency and communication.

User Acceptance Testing (UAT) UAT is performed in a user environment that resembles the production environment. It verifies that the delivered system meets the user's requirements and is ready for use in real time.

Advantages of V-Model

Suited for Restricted Projects

Due to the stringent nature of the V-Model and its linear design, implementation, and testing phases, it's perhaps no wonder that V-Model has been heavily adopted by the medical device industry in recent years. In situations where the project length and scope are well-defined and limited, the technology is stable, and the documentation and design specifications are clear and constant, then this will be a great method.

Ideal for Time Management

Along the same vein, V-Model is also well-suited for projects that must maintain a strict deadline and meet key milestone dates throughout the process. With fairly clear and well-understood stages that the whole team can easily comprehend and prepare for, it is relatively simple to create a timeline for the entire development lifecycle, while generating milestones for each stage along the way. Of course, V-Model in no way ensures milestones will always be met, but the strict nature of the model itself enforces the need to keep to a fairly right schedule.

Disadvantages of V-Model

Lacks Adaptability

Similar to the issues facing the traditional waterfall model on which the V-Model is based, the most problematic aspect to the V-Model is its inability to adapt to any necessary changes during the development lifecycle. For example, an overlooked issue within some fundamental system design, that is then only discovered during the implementation phase, can present a severe setback in terms of lost man-hours as well as increased costs.

Timeline Restrictions

While not an inherent problem with the V-Model itself, the focus on testing at the end of the life cycle means that it's all too easy to be pigeonholed at the end of the project into performing tests in a rushed manner to meet a particular deadline or milestone.

Ill-Suited for Lengthy Life Cycles

Like the waterfall model, the V-Model is completely linear and thus projects cannot be easily altered once the development train has left the station. V-Model is therefore poorly suited to handle long-term projects that may require many versions or constant updates/patches.

Encourages 'Design-by-Committee' Development

While V-Model is certainly not the only development model to fall under this criticism, it cannot be denied that the strict and methodical nature of the V-Model and its various linear stages tend to emphasize a development cycle befitting managers and users, rather than developers and designers. With a method like V-Model, it can be all too easy for project managers or others to overlook the vast complexities of software development in favor of trying to meet deadlines or to simply feel overly confident in the process or current progress, based solely on what stage in the life cycle is actively being developed.