



Software Processes

V-Model

Adaptive Software Development

Software Engineering 2018

Bruno Carvalho
up201606517

Diogo Yaguas
up201606165

Tiago Castro
up201606186

October 25, 2018

Contents

1	Adaptive Software Development	2
1.1	The Process of Adaptive Software Development	2
1.1.1	Speculation	2
1.1.2	Collaboration	3
1.1.3	Learning	3
1.2	Artifacts	3
1.3	Pros and Cons	4
2	V-Model	5
2.1	The Process of V-Model	5
2.1.1	Design Phase	5
2.1.2	Testing Phases	6
2.2	Advantages of V-Model	6
2.2.1	Suited for Restricted Projects	6
2.2.2	Ideal for Time Management	6
2.3	Disadvantages of V-Model	7
2.3.1	Lacks Adaptability	7
2.3.2	Timeline Restrictions	7
2.3.3	Ill-Suited for Lengthy Life Cycles	7
2.3.4	Encourages ‘Design-by-Committee’ Development	7

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.

ASD derives from a conceptual perspective based on the Complex Adaptive Systems (CAS) theory and a practical perspective that supports itself on the Rapid Application Development techniques and deterministic methodologies. From CAS, it inherits its Emergence and Complexity.

The ASD cycle has six basic characteristics:

1. Mission focused
2. Feature based
3. Iterative
4. Time-boxed
5. Risk driven
6. Change tolerant

The Process of Adaptive Software Development

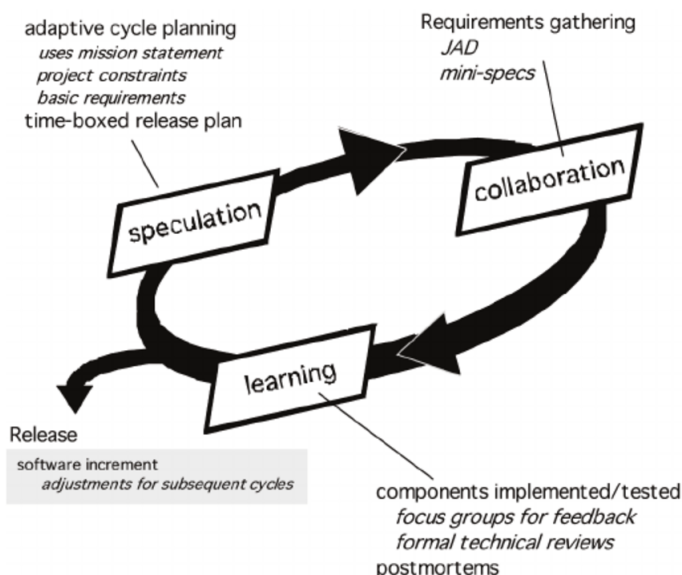


Figure 1: The Adaptive cycle

Adaptive replaces the traditional waterfall model 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 in the various phases.

1. Speculation

This is the setup phase. During it, developers attempt to understand the exact nature of the product's requirements — what features need to be added, what problems need to be fixed, and what changes need to be made.

To facilitate this, a previous cycle feeds information to this phase, namely bugs, user reports, new requirements found during developments and finished or unfinished features.

Let's see how *speculation* comprises *project initiation* and *risk-driven cycle planning*.

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

Risk-Driven Cycle Planning The mission statement's aim is to expose the customer's desired result for the project. It's likely that 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.

2. Collaboration

This is the development phase. Here, effective collaboration between teams is key. Complex applications are not built — they evolve. In this high information flow environment, one person or small group can't possibly *know it all*, so collaboration skills are paramount.

Collaboration with the client remains to handle minor issues or requirements adjustments, and preference resolutions.

3. Learning

This is the quality control phase. Here is where many integration and system tests are performed. Knowledge is gathered about the state of the project to prepare for the next speculation phase.

Focus groups, technical reviews and project postmortems are conducted in this phase. Giving a status report to the customer and receiving feedback is the common goal. In a focus group, the application or system being developed is presented to the customer, who then gives feedback, records change requests, and helps prepare a mission statement.

Summing up, there are four general categories of things to learn about at the end of each iteration:

- Result quality from the customer's perspective.
- Result quality from a technical perspective.
- The functioning of the delivery team and the practices team members are utilizing.
- The project's status.

Artifacts

Project Vision A document which serves to establish a focus for the project, and identifies the foundation on which to build the team's commitment. It should help developers explain the project's purpose within a few minutes. The document may include: Project Background; Project Scope; Sponsors; Customers; Business Functional Objectives; Project Risks; Staffing Requirements; Constraints; Assumption...

Project Data Sheet A document which results from a project initiation. Includes: client, project objective statement, features, performance or quality attributes, architecture, issues and risks, milestones, and the team's core members.

Product Mission Profile A short document used to record focus for the next development iteration, using for instance profile matrices (Figure 2). These may be done per iteration or per feature. In the example, the *Excel* column identifies Scope as the most important dimension for the next iteration.

Product Quality Dimensions	Priority Level		
	Excel	Improve	Accept
Scope (Features)	◆		
Schedule		◆	
Defects			◆
Resources			◆

Project Specification Outline It serves several purposes. First, it provides the stakeholders and team members with a reasonable understanding of the boundaries and scope of the development effort. Second, it is a baseline for size estimation. Finally, the PSO facilitates adaptive cycle planning and is accomplished by assigning product features to specific cycles. The specification outline's primary objective is to define the features or functionality of the product.

Figure 2: Mission Profile's Matrix

Pros and Cons

Pros

- Allows for transparency because developers explain the state of the project to the client before, during and after each stage;
- Generates high quality software, breaking down the project into components, concentrating on high quality development, testing of one component at a time, aids in producing a high quality end product in the end run.

Cons

- Requires a lot of the client's insight, thus requiring a lot of time from them;
- Due to the lack of predictability, it is hard to develop the business case for the project and makes it even more difficult to negotiate for fixed prices and cost of the projects;
- Testing at every stage helps deliver a quality product but increases the cost of the project in the long run and brings a lot of projects to failure.

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

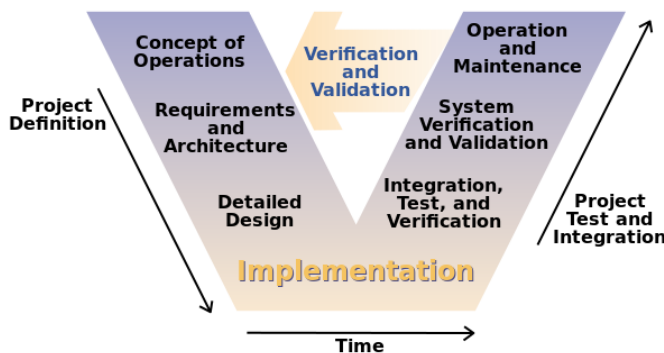


Figure 3: The V-Model stages

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 is the 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. 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.

References

- [1] Jim Highsmith. *Agile Software Development Ecosystems*. Addison-Wesley Professional, Apr. 2002, pp. 173–179.
- [2] Andrew Powell-Morse. *V-Model: What is It And How Do You Use It?* URL. Dec. 2016. (Visited on 10/24/2018).
- [3] Margaret Rouse. *What is V-Model (Vee-Model)?* URL. June 2016. (Visited on 10/24/2018).
- [4] www.geeksforgeeks.org. *Software Engineering | SDLC V-Model*. URL. (Visited on 10/24/2018).