WHITE PAPER

Systems Engineering and Development

Experts discuss requirements management, the V-Model, and verification and validation

Across industries, one of the trends that's increasingly apparent is the rapidly growing complexity of systems, organizations, processes, and supply chains.

Systems engineers—those who have interdisciplinary training to successfully orchestrate and manage complex systems—are uniquely suited to overcome the new challenges being created from this emerging landscape.

We believe systems engineering will play a larger role in product development moving forward, so we asked a number of experts for their thoughts on the interdisciplinary approach and product development.

In this white paper, you'll find:

We hope you find these essays informative and capable of sparking new conversations as product development moves forward.

1. The Reliance on Requirements Management Tools as the Engineering Messiah

BY CHRISTER FRÖLING

I've met a lot of folks who have an unhealthy belief in requirements management as the sum total of requirements work needed.

For example, whenever I ask if someone works with requirements, almost immediately the conversation turns into two things: either a discussion on tooling (IT support in the shape of a relational database for tracing different requirement statements to and from each other) or a conversation around the need to change to more Agile ways of working using methods like user stories, personas, modeling, etc.

The thing is: I asked if someone works with requirements. Not about tools or development processes or methods.

What Are Requirements?

Requirements are capabilities that a product must meet to satisfy a stakeholder's need to solve a specific problem. The stakeholder's needs can come from a number of sources, including compliance to a standard or regulation, a business need, a technical situation, market need, competition, etc.

So, why are so many discussions about tooling or Agile methods?

My sense is that:

A: People don't know what requirements are really used for

OR

B: They don't understand why they work with requirements in the first place

The definition of the discipline of requirements engineering—all things associated with requirements—is:

> Requirements engineering (RE) refers to the process of defining, documenting, and maintaining requirements in the engineering design process."

So, when I put requirements and process together, I end up with the notion of being able to understand a need and/or problem, defining it in a suitable way (notation technique, language, and "chunks"), and maintaining this clarity throughout the project to solve the specific need and problem.

That's the boiled-down version.

That means to get from point A (the problem) to point B (the solution), I need to do a lot of clever stuff to understand, define, build, prove, and deliver B to satisfy A.

The way of working (that endless discussion about development methodologies) is of lesser importance here. You still need to validate in some way that you have understood the problem before you start building a solution.

To solve complex things, I need to develop suitable pieces of this puzzle and give these to the people helping me. To do this, I typically need clear pieces of information that, together with the other components, complete the picture.

I need to describe this specific, correct, and unambiguous piece of information. This basically becomes a tree-like shape, with the original need at the top, which is broken down and developed into those well-known requirement types of functions, constraints, and capabilities we are used to seeing.

Each broken-down requirement is derived to satisfy its originator one branch up in the hierarchy. For this, we need to keep control over the links that become one of the results of this activity.

Otherwise, how can we know what the consequence of a change will be?

So, back to the question on a requirements management tool. Is it enough to have a tool for managing those requirements and links?

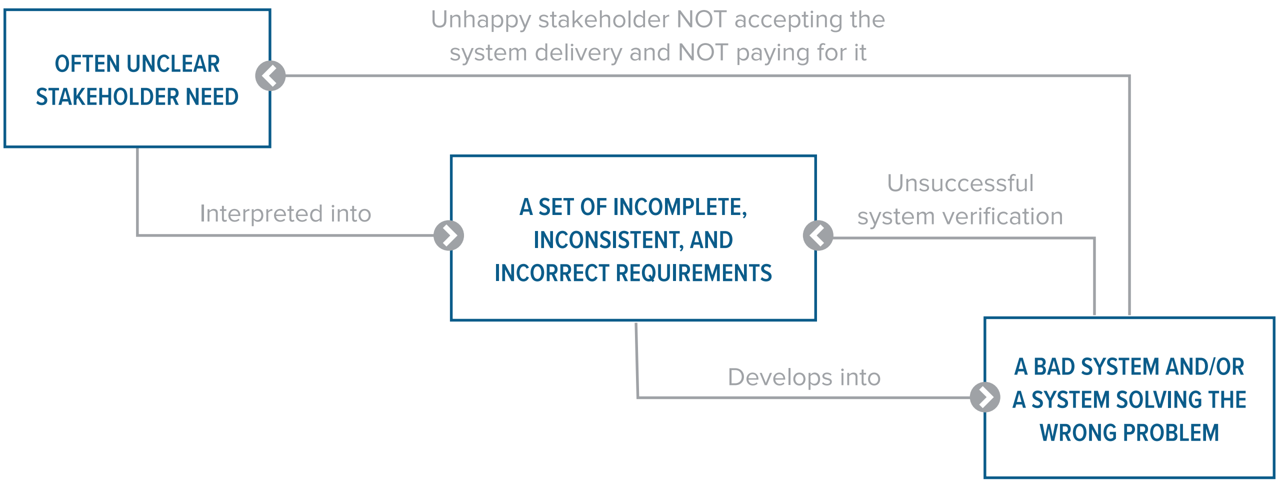
No, absolutely not! Have you heard the phrase "garbage in, garbage out"?

If you focus only on the management part of requirements engineering, you'll create an overreliance on the tool to show when a link requires review due to a change somewhere.

Haven't you forgotten something?

Requirements are stated by humans for humans, and the knowledge captured because of a decision to state a new requirement in the first place must be captured in the original sentence.

A badly formulated requirement is the parent of a lot of new requirements, with all of them inheriting the bad genes:



Unhappy stakeholder NOT accepting the system delivery and NOT paying for it
OFTEN UNCLEAR STAKEHOLDER NEED
Interpreted into
A SET OF INCOMPLETE, INCONSISTENT, AND INCORRECT REQUIREMENTS
Unsuccessful system verification
Develops into
A BAD SYSTEM AND/OR A SYSTEM SOLVING THE WRONG PROBLEM

Figure 1.

Why Requirements Quality is Important

Don't get me wrong. Requirements management IS important. I agree 100 percent.

One needs to be able to trace requirements to do impact analysis on suggested changes, do testing in a clever and incremental way, and basically have an idea when to be done with the project, but…

Wrong information derived from and traced into more low-quality information will lead to disaster.

A recent study showed that of all failed projects investigated, more than 40 percent concluded that failure was due to bad requirements or the inability to understand the stakeholder's true need in the first place.

> If you are lucky, your project won't fail due to bad requirements. Maybe you'll "only" get a lot of late design changes, angry test teams, frustrated project managers, failed financial goals, and unhappy customers.

You better stop saying that you are good at requirements solely because of the fact that you own a requirements management tool!

Without good engineering practices to discover and formulate correct and consistent requirements and making that specification complete (to describe in full the functions, qualities, and constraints), you will still have huge problems, requirements management tool or no. "Garbage in" is, and will always be, "garbage out."

PS: The people in that study who didn't blame the requirements for their recent project failure blamed the project manager. Just so you know. The story continues, and so will the failed projects.

ABOUT THE AUTHOR

Senior Consultant and Sales Manager North & East Europe for the REUSE Company, Christer Fröling is a systems engineer with a focus on "holistic" Systems Engineering and its specialty engineering disciplines and has more than 20 years of experience. Check out his blog at requirementdoctor.blog.

2.
Myths About the V-Model

BY MICHAEL JASTRAM

If you are building complex systems, then you have probably heard of the V-Model. Before we explain it, a word of caution: There are different, partly contradictory, definitions of the V-Model. Further, it's easily misunderstood, which can be more harmful than helpful. This is also the source of the myths of the V-Model. Understanding the myths about the V-Model helps you apply it correctly, thereby making you more effective.

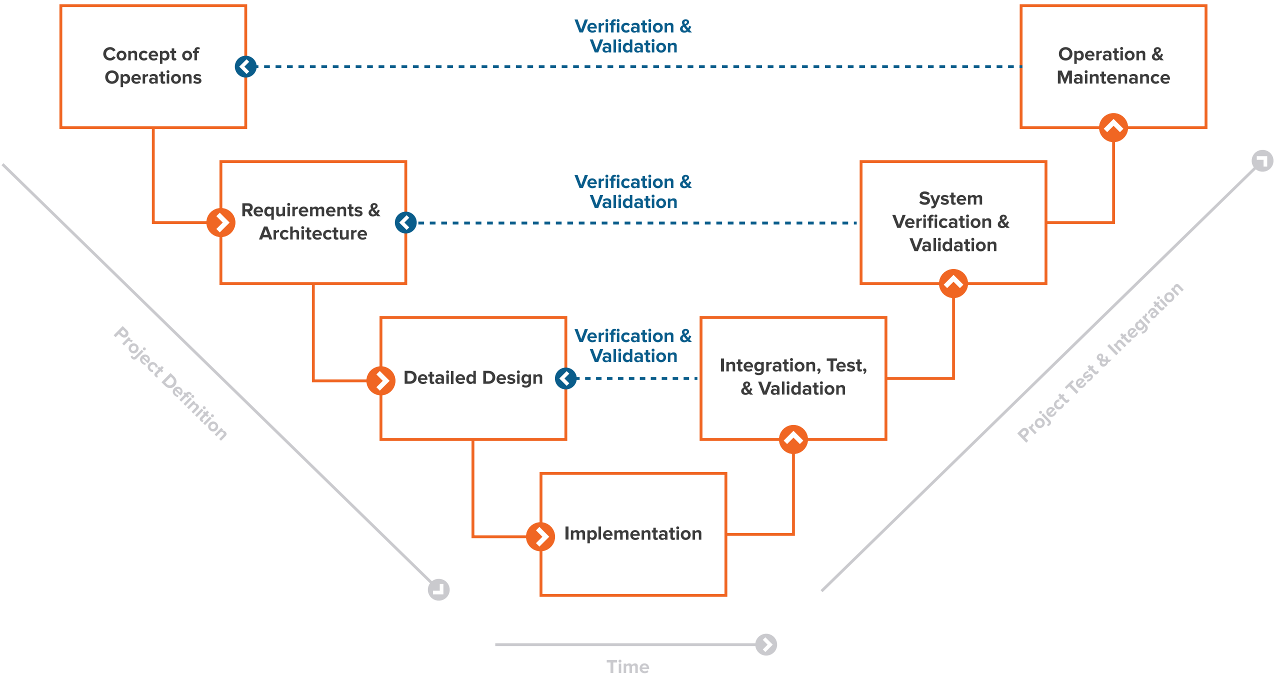

Concept of Operations
Verification & Validation
Operation & Maintenance
Requirements & Architecture
Verification & Validation
System Verification & Validation
Detailed Design
Verification & Validation
Integration, Test, & Validation
Implementation
Project Definition
Project Test & Integration
Time

Figure 2. The V-Model in Brief

Above is one visualization of the V-Model. To understand it, we start in the top left. Here you find the high-level requirements ("concept of operations"). Following the left arm of the V, the product definition is refined until we reach the implementation. The right arm of the V contains the corresponding items for validation and verification, like tests, analysis, or inspection.

What's unclear about the model (and frequently a source of confusion) are the relationships between the elements, visualized through arrows. Note that the various arrows have different meanings:

The solid lines connecting the boxes represent dependencies. For instance, there is a relationship between "requirements" and "detailed design" (the detailed design is a possible solution). If the requirement changes, then usually the design needs to change as well. For instance, changing the battery life from 24 to 48 hours (requirement) will probably necessitate a larger battery (design).

The dashed "verification and validation" lines also represent a dependency, but this time for test coverage. Note that the arrows in the figure represent a "tests" relationship ("system verification and validation tests requirements and architecture"). The direction of the dependency is in the other direction (e.g., changing the battery life from 24 to 48 hours will trigger an update to the corresponding systems test).

The outer solid lines denote activities, and broadly represent time. But here we have to be very careful because they do not represent a strict time sequence. This is a huge source of confusion.

Now that we have a rough understanding of the V-Model, let's look at four of its common myths:

Myth: The V-Model is Only an Extended Waterfall

If we take the time axis too literally, then the V-Model appears to be the Waterfall Model (the left side of the V), with the verification "foldeded up." Just as a reminder, the Waterfall Model is a linear development approach. It consists of different phases (requirements, design, implementation, etc.), where each one is completed before the next starts. The main difference between the V-Model and Waterfall is the fact that the V-Model iteratively increases the maturity of the development, affecting all items of the product description.

Myth: The V-Model is Simply a Process

This statement is correct, but not complete. Yes, the V-Model is a process. In fact, it is widely used, cited, and standardized in various ways: There is the German "Das V-Modell," a government standard, and the PRINCE2 method, which is used in the UK and US. Many standards reference it, including ISO/IEC/IEEE 15288.

But this view focuses on the process and ignores the fact that the V-Model also represents the artifacts of product development and their relationships. This second view can generate additional insights, especially if the artifacts are fine-grained. For instance, traditionally the “requirements” box in Figure 2 would represent a requirements process, with a process flow to the design process.

But if these represent individual requirements, with a precise traceability to design elements, this relationship can suddenly be used for coverage analysis (“ensure all requirements are covered”), change management (“after this requirement change, which design elements are affected?”), and much more.

Looking at both—process and data model—significantly increases the value of the V-Model.

Myth: The V-Model Cannot be Applied in an Agile Environment

This myth is related to the previous one: The existing methods based on the V-Model were created in the ’90s, before the Agile Manifesto.

But the V-Model itself is not just compatible with Agile methods—it’s (almost) a prerequisite! To be more precise, a robust relationship model, like the V-Model, is the foundation for making changes with confidence. And frequent changes are one of the aspects of Agile development. A robust relationship model reflects the triangular relationship between requirement, test, and implementation.

Myth: During Development, We Move From Left to Right

As already indicated in the prior three myths we’ve busted, the V-Model requires jumping back and forth during development. Nevertheless, the assumption is that we need requirements before we can create the corresponding tests.

But the V-Model is well suited to also support behavior-driven development (BDD). In BDD, the expected behavior is defined in a testable manner. Design only starts after the behavior is defined.

This corresponds to test-driven software development, where an automated test is written before the software implementation is created. For BDD to work, the relationships between the items concerned must be clearly defined and available for analysis and change management.

The V-Model provides the necessary structures for this process. In fact, this triangular relationship is visible in the V-Model’s triangular form, except that BDD uses a slightly different terminology.

The V-Model and Product Development Platforms

The V-Model can be confusing, as it relates to both a process and a data model. Solutions like the Jama Product Development Platform distinguish between these two aspects and address them separately in the following ways:

- Work with a flexible data model consisting of item types and relationships. These can be tailored to existing artifacts and workflows. This allows the product description to evolve naturally, while managing the relationships automatically. The results are a clearer understanding of coverage without the fear of changes.
- Allows workflows to guide the maturity of the product description, aligned with existing processes. This creates the confidence in the quality of the product, makes it easy to work in a compliant environment, and takes the anxiety out of audits.

The V-Model is a powerful concept. But it has been around for a long time: It was created in a time when document-based work was the norm and Agile was not yet an established approach. Applied in a modern context, the V-Model is the enabler for the faster development of better products. But in order to take advantage of this, it must be understood.

ABOUT THE AUTHOR

Dr. Michael Jastram is a systems engineer with a focus on requirements modeling. He is General Manager of Formal Mind GmbH and operates ReqIF.academy, an online library for requirements exchange knowledge. He further supports Jama Software as a Senior Solutions Architect.

3. The Difference Between Verification and Validation

BY LOUIS S. WHEATCRAFT

One question I hear often is, “What is the difference between verification and validation?”

In general, verification refers to the basic structure of an item (requirements, design, system) being verified. Verification ensures an item meets requirements that drive its creation, whether it be rules on writing well-formed requirements; standards and best practices (external and internal) on the design; or requirements on the coding or manufacturing of the system.

Validation, on the other hand, extends beyond the basic structure into how well the item communicates or addresses stakeholder needs and expectations while functioning in the intended operational environment.

For context, in this article, my use of the word *system* refers to the system of interest — regardless of the level on which the system of interest exists within the architecture (system, subsystem, assembly, component).

Variance of Terminology

Requirements are developed during the concept phase and represent a conceptual view of the system of interest.

The transformation of the conceptual view of the system as communicated by the baseline requirement set into a physical realization of the system is referred to as design. This process evolves from a preliminary design, to a final design, and then to development, which transforms the design into the physical system (hardware) or code (software).

While the terms *verification* and *validation* are commonly used, the true meaning of the concepts represented in each are often misunderstood. Thus, the terms are used interchangeably without making clear the context in which they are used — resulting in ambiguity.

Within a single organization, whenever engineers are asked to define these terms, you will hear different, and often conflicting, definitions.

To avoid this ambiguity, each term needs to be preceded by a modifier (i.e., the subject), which denotes the proper context in which the term is being used. Specifically:

- Requirement verification or requirement validation
- Design verification or design validation
- System verification or system validation (as shown in Figure 3)

The concepts of verification and validation are very different depending on the modifier. When using these terms, it should be clear as to which concept is intended.

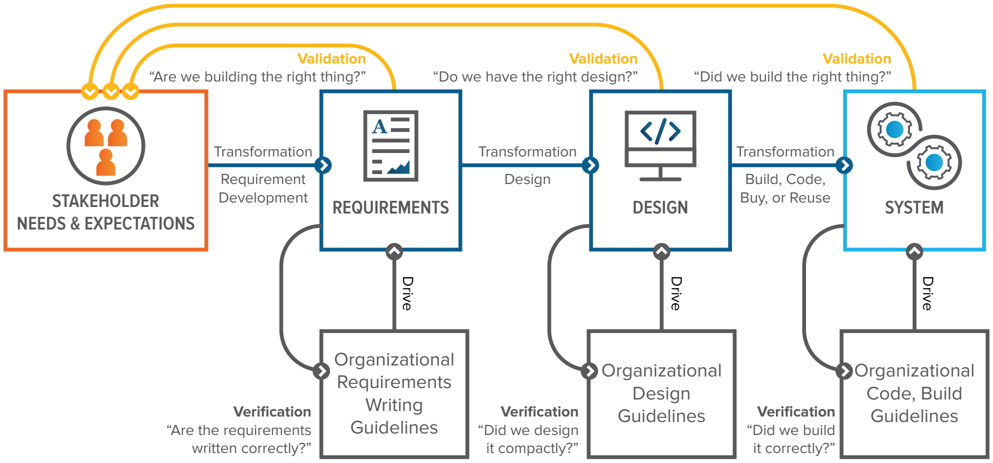

Validation
"Are we building the right thing?"
Validation
"Do we have the right design?"
Validation
"Did we build the right thing?"
STAKEHOLDER NEEDS & EXPECTATIONS
Transformation
Requirement Development
REQUIREMENTS
Transformation
Design
DESIGN
Transformation
Build, Code, Buy, or Reuse
SYSTEM
Drive
Drive
Drive
Organizational Requirements Writing Guidelines
Organizational Design Guidelines
Organizational Code, Build Guidelines
Verification
"Are the requirements written correctly?"
Verification
"Did we design it compactly?"
Verification
"Did we build it correctly?"

Figure 3: Verification and validation are the processes of confirming that artifacts generated during the transformation processes are acceptable.

Example

As shown in Figure 3, a requirement set results from a formal transformation of stakeholder needs and expectations. Correspondingly, design is a result of a formal transformation of the requirement set into an agreed-upon design, and a system is a formal transformation of that design into a system.

The process of creating a requirement set involves:

- Analyzing stakeholder needs and expectations to obtain the necessary elements to be included in the requirement set.
- Identifying the characteristics of the desired result against the organizational guidelines and rules by which the requirement statements and requirement set is to be written.
- Transforming the stakeholder needs and expectations into a set of requirements that unambiguously communicates the stakeholder needs and expectations to the design organization (requirement validation) and conforms to the organization's guidelines for writing well-formed requirements and requirement sets (requirement verification).

Requirement Verification and Validation Defined

In this context, requirement verification confirms, by inspection, that the requirements contain the necessary elements and characteristics of a well-formed requirement. It also ensures that the requirement set conforms to the rules set forth in the organization's requirement development guidelines.

Requirement validation confirms, by inspection and analysis, that the resulting requirement set meets the intent of the stakeholder's needs from which the requirements and requirement set were decomposed or derived. Thus, the requirement statements and requirement set are confirmed by both verification and validation activities.

Based on this discussion, to help remove the ambiguity in the use of the terms *verification* and *validation*, the following definitions of these terms are included within the context of a product lifecycle:

- **Requirement Verification**
  The process of ensuring the requirement meets the rules and characteristics defined for writing a good requirement. The focus is on the wording and structure of the requirement.

  For example: Is the requirement worded or structured correctly in accordance with the organization's standards, guidelines, rules, and checklists?

- **Requirement Validation**
  Confirmation that the requirements and requirement set is an agreed-upon transformation that clearly communicates the baselined stakeholder's needs and expectations in a language understood by the developers. The focus is on the message the requirements and requirement set is communicating.

  For example: Do the requirements and requirement set clearly and correctly communicate the baselined stakeholder expectations and needs?, Are we doing the right things? Are we building the right thing [as defined by the requirement set]?

Requirement verification and requirement validation activities should be done continuously as the requirements are developed at each level of the architecture.

It should also be part of baseline activities of the requirement set performed during the System Requirements Review (SRR) or a similar type of gate review at each level.

Note: Most organizations do not make a distinction between requirement verification and requirement validation. Rather, they use only the phrase *requirement validation* to mean both. Using the phrase *requirement verification* often confuses the conversation because many interpret *requirement verification* to have the same meaning as *system verification*, as defined in the article's next section.

System Verification and Validation Defined

Once the design is baselined, it's also transformed — via build, code, buy, or reuse — into the system of interest. Similar to the discussion for the design process, most organizations have a set of guidelines or "golden rules" that instruct the build (manufacture or code) process. These include workmanship and quality control requirements for the organization.

After the system has been built or coded, there will be a gate review wherein the system is both verified and validated. At this stage of the system lifecycle, the concepts of system verification and system validation take on a more formal meaning. Thus, the systems engineering (SE) lifecycle processes include the processes of system verification and system validation.

Each process represents a set of activities (test, demonstration, inspection, analysis) that cumulate with one or more gate reviews associated with the acceptance of the system by the customer.

Thus, **system verification** is a formal SE process and has a legal aspect where that developer is proving the system reflects the baselined requirements have been met. From a contracting perspective, the baselined requirements are a type of contract and are legally binding.

In this context, system verification has two aspects:

- Does the built or coded system of interest clearly represent the requirements that drove the design? Did we build the right thing?
- Did the build or code team follow the organization's guidelines for manufacturing and coding?

Following system verification, system validation is performed. Again, **system validation** is a formal SE process and has a legal aspect. In this process, the developer is proving whether or not the built (or coded) and verified system results in the intended purpose being met in the operational environment. The developer is also ensuring the stakeholder's expectations and needs are being met.

Like the system requirements, the baselined stakeholder's needs and requirements defined during the scope definition phase can also be considered part of a contract and are legally binding.

Based on this discussion, to help remove the ambiguity in the use of the terms *verification* and *validation*, the following definitions for system verification and system validation are included in terms of a product lifecycle context.

System Verification

A process performed after design and build or coding, making sure the designed and built or coded system meets its requirements. The focus is on the built or coded system and how well it meets the agreed-to requirement set that drove the design and fabrication. Methods used for system verification include: test, demonstration, inspection, or analysis. "Did we build the thing right?"

Also included in system verification is a determination that the team responsible for building or coding the system of interest followed the organization's rules, guidelines, and best practices associated with manufacturing and coding. The focus is on the manufacturing or coding processes. "Did we follow our organization's guidelines for manufacturing or coding correctly?"

System Validation

A process that occurs after system verification that makes sure the designed, built, and verified system meets its intended purpose in its operational environment.

The focus is on the completed system and how well it meets stakeholder expectations (needs) that were defined during the scope definition phase that should have occurred at the beginning of the project. “Did we build the right thing?”

System verification and system validation processes are directly related to the contractual obligation concept for a requirement statement and set of requirements.

It is through these process activities that we prove we have met both the agreed-to requirements and the agreed-to needs of the entities who are the source of or own them. This is often accomplished as part of certification and acceptance activities.

ABOUT THE AUTHOR

Lou Wheatcraft is a consultant and trainer for Requirements Experts/Seilevel as an expert in requirements development and management from a systems engineering perspective. Lou is Chair of the INCOSE Requirements Working Group and author of a series of blogs which can be found at www.reqexperts.com/blog.

ABOUT JAMA SOFTWARE