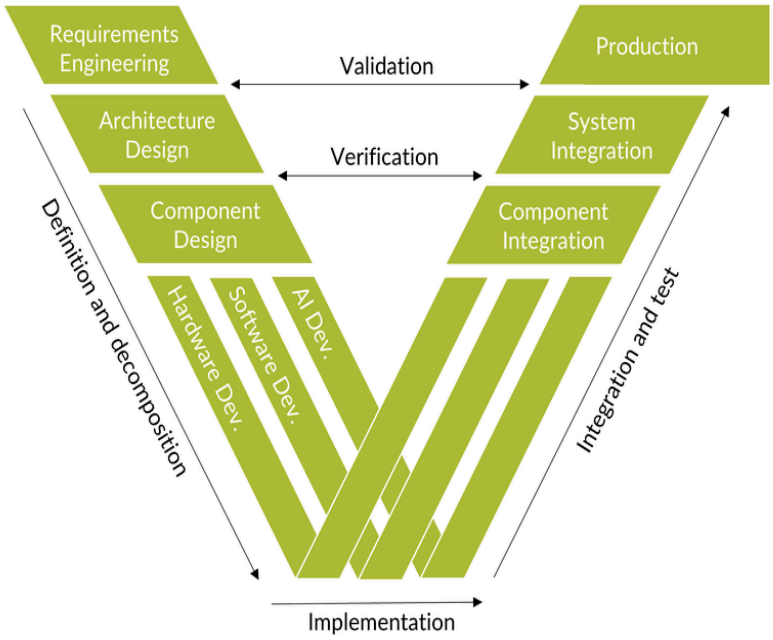
The V-Model is a software development method often found in areas with high requirements on safety and security, which are common in highly regulated areas. Combining the traditional V-Model with a disciplined agile approach promises to allow as much agility as possible, while addressing the issues often found in AIoT initiatives: complex dependencies, different speeds of development, and the "first time right" requirements of those parts of the system which cannot be updated after the Start of Production (SOP).

Recap: The V-Model

The V-model is a systems development lifecycle which has verification and validation "built in". It is often used for the development of mission critical systems, e.g., in automotive, aviation, energy and military applications. It also tends to be used in hardware-centric domains. Not surprisingly, the V-model uses a v-shaped visual representation, where the left side of the "V" represents the decomposition of requirements, as well as the creation of system specifications ("definition and decomposition"). The right side of the "V" represents the integration and testing of components. Moving up on the right side, testing usually starts with the basic verification (e.g., unit tests, then integration tests), followed by validation (e.g., user acceptance tests).



Advantages

- Well suited for stabilizing complex requirements, e.g. required by multi-company development projects
- Built-in verification and validation, which is important for highly regulated / mission critical applications

Disadvantages

- Inherently waterfall
- Very rigid and inflexible
- Does not support iterative / incremental development: No early prototypes, MVP, continuous improvement
- Not possible to adapt to changes happening midway in the development
- Sometimes seen as having a bad reputation

aiotplaybook.org

When applying the V-model to AIoT, it needs to take different dimensions into consideration; usually including hardware, software, AI, and networking. In addition to the standard verification tests (unit tests, integration

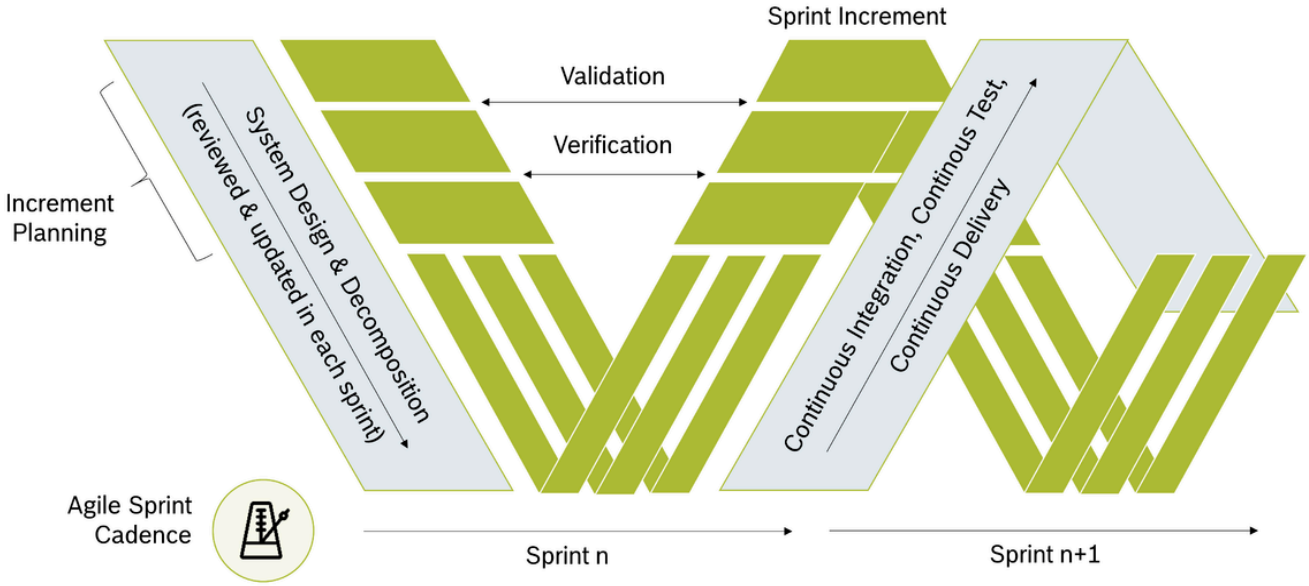
tests) and validation tests (user acceptance and usability tests), the V-model for AIoT also needs to address interoperability testing, performance testing, scalability testing and reliability testing. The highly distributed nature of AIoT systems will pose specific challenges here.

Test automation is key to ensure a high level of test efficiency and test coverage. On the software side, there are many tools and techniques available to support this. In the AI-world, these kinds of tools and techniques are only just beginning to emerge, which makes it likely that a more custom approach will be required. In the embedded and hardware world, simulation techniques such as Hardware-in-the-Loop (HIL), Software-in-the-Loop (SIL) and Model-in-the-Loop (MIL) are well established. However, most AIoT products will also require testing of the actual physical product and how well they perform in the field in different types of environments. Again, this will be a challenge, and some ingenuity will be required to automate testing of physical products wherever possible.

Evolution: The Agile V-Model

The AIoT framework aims to strike a good balance between the agile software world and the less agile world of often safety-critical, complex and large(r)-scale AIoT product development with hardware and potentially manufacturing elements. Therefore, it is important to understand how an agile method works well together with a V+V-centric approach such as the V-model. The logical consequence is the *Agile V-model*. Combining agile development with the V-model is not a contradiction. They can both work very well together, as shown in the figure following:

- Agile methods use story maps including epics, themes, features and user stories for logical decomposition. This maps well to the left side of the V
- Continuous Integration / Continuous Test / Continuous Delivery are inherently agile methods, that map well to the right side of the V
- The key assumption is that the V-model is not used like one large waterfall approach. Instead, the Agile V-model must ensure that the sprints themselves will become Vs according to the V-model



There are two options to implement the latter:

- Each sprint becomes a complete V, including development and integration/test
- The agile schedule introduces the concept of dedicated integration sprints.

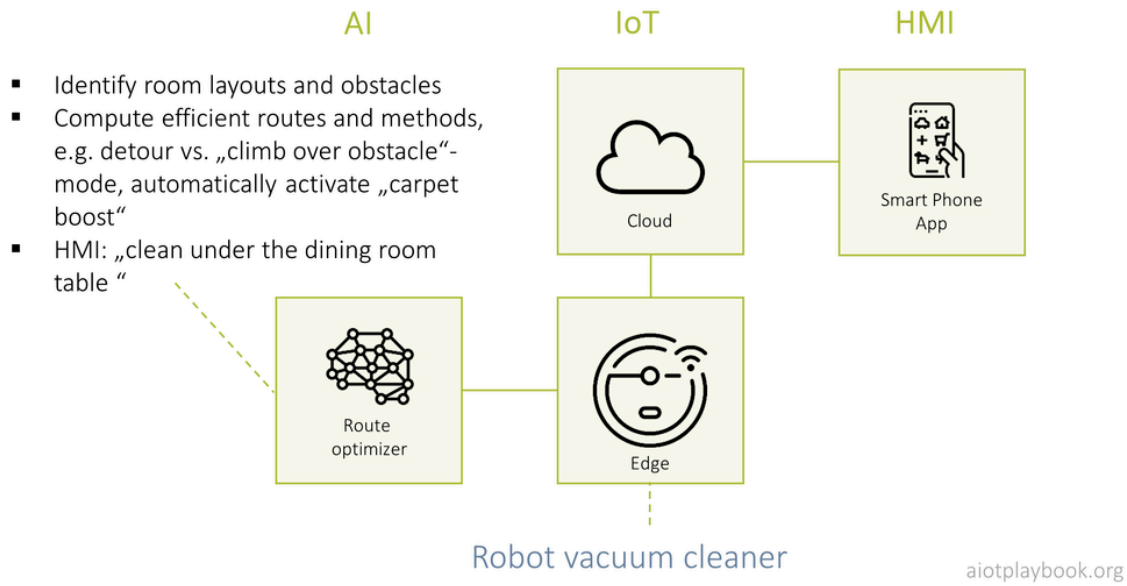
- One V becomes 2 sprints: one development sprint, one integration sprint
- There are pros and cons to both approaches
- The complexity and scale of the project will surely play a role in determining the best setup

For most projects / product teams, it is recommended that development and integration are combined in a single sprint ("v-sprint"). Only for projects with a very high level of complexity and dependencies, e.g., between components developed by different organizations, is it recommended to alternate between development and integration sprints. The latter approach is likely to add inefficiencies to the development process, but could be the only approach to effectively deal with alignment across organizational boundaries.

The ACME:Vac vacuum robot example

To illustrate the use of the Agile V-Model, the realistic yet fictitious ACME:Vac example is introduced. This is a robot vacuum cleaning system that combines a smart, connected vacuum robot with a cloud-based back end, as well as a smart app for control.

Modern robot vacuum cleaners are very intelligent, connected products. Even the most basic versions provide collision, wheel, brush and cliff sensors. More advanced versions use visual sensors combined with a VSLAM algorithm (Visual Simultaneous Location and Mapping). The optical system can identify landmarks on the ceiling, as well as judge the distance between walls. The most advanced systems utilize LIDAR technology (Light Detection and Ranging) to map out their environment, identify room layouts and obstacles, and serve as input for computing efficient routes and cleaning methods. For example, the robot can decide to make a detour vs. switching into the built-in "climb over obstacle"-mode. Another example is the automatic activation of a "carpet boost" mode. IoT-connectivity to the cloud enables integration with user interface technology such as smart mobile devices or smart home appliances for voice control ("clean under the dining room table"). Edge AI algorithms deployed on the robot are used to control these processes in advanced models. A complete design of the ACME:Vac is provided in the Product / Solution Design section.

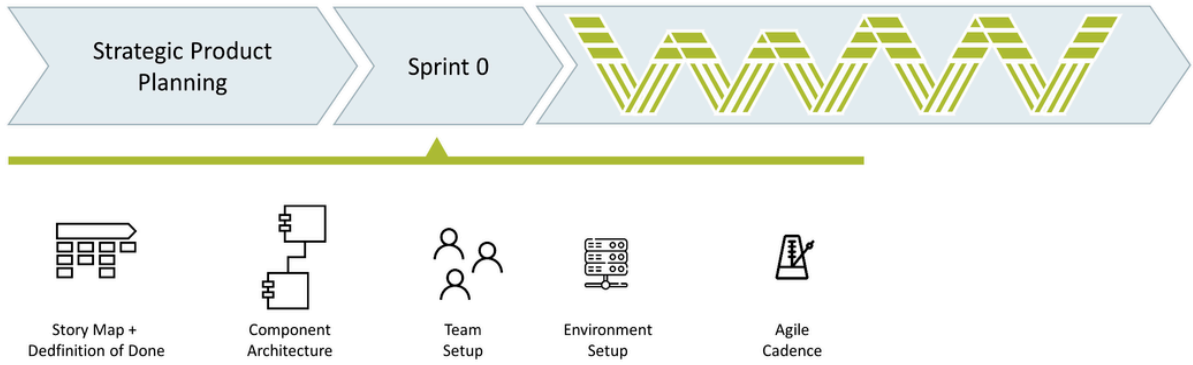


Applying the Agile V-Model to ACME:Vac

The following provides a description of how the ACME:Vac example is developed using the Agile V-Model. This discussion will start with a look at a "Sprint 0", where the foundations are laid. Then a "Sprint n" is discussed in detail, with an outlook on the transition to the next sprint ("Sprint n+").

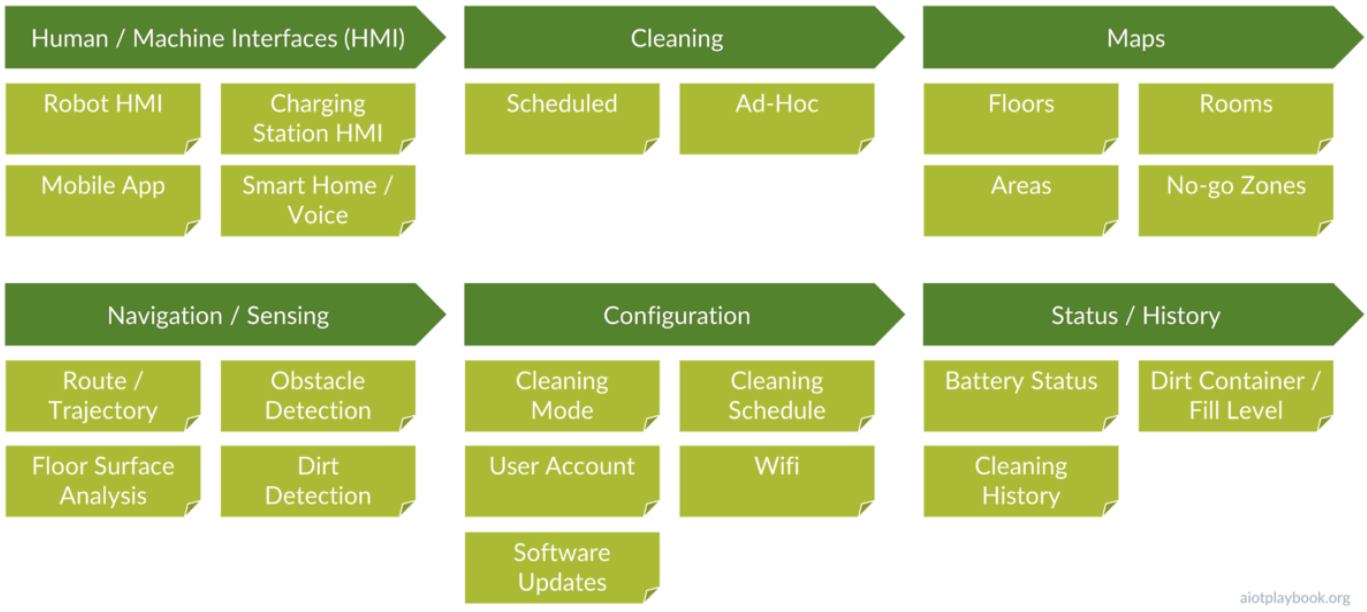
Sprint 0

Many scrum masters are starting their projects with a "Sprint 0" as a preparation sprint to initiate the project (ignoring the quasi-religious discussion whether this is a good practice or a "cardinal sin" (https://mdalmijn.com/why-using-sprint-0-is-a-cardinal-sin-for-all-scrum-masters/) for the moment, since we have to start somewhere...). In the case of ACME:Vac, two working results arise: the initial story map and the initial component architecture. These will be important elements for the planning of the subsequent sprints.



Initial Story Map

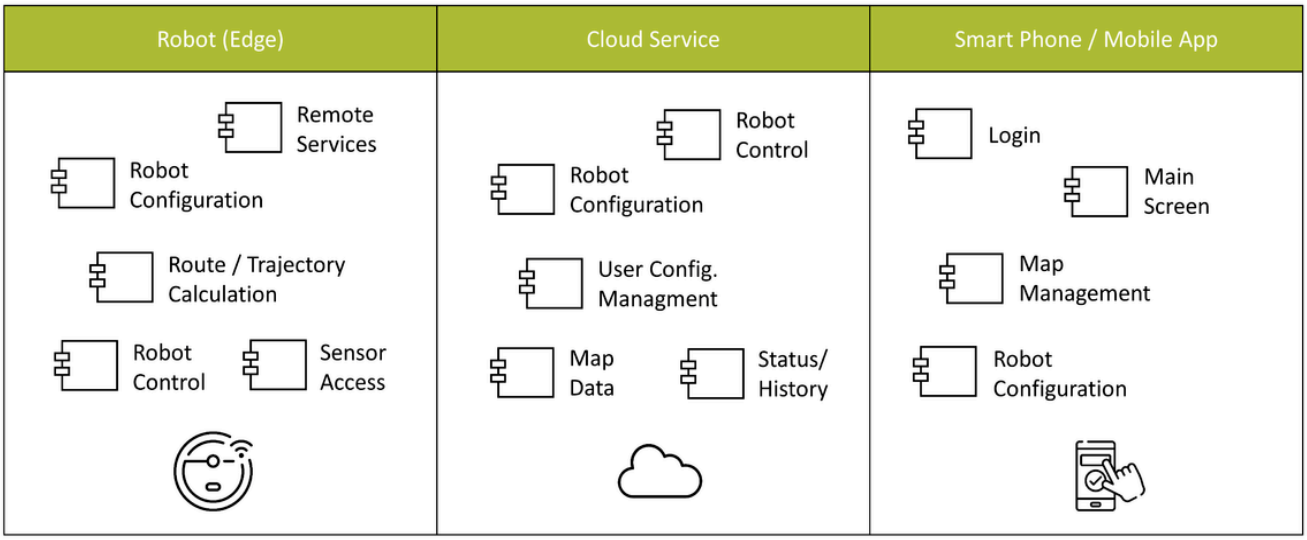
According to the story map structure proposed by the AIoT Framework, the story map for the vacuum robot system includes epics and features on the top level. The epics include Human/Machine Interfaces, the actual cleaning functions, management of the maps for the areas to be cleaned, navigation/sensing, system configuration, and status/history.



Initial Component Architecture

The Component Architecture for the ACME:Vac highlights the key functional components in three clusters: the robot itself (Edge), the cloud back end, and the smartphone app. On the robot, two embedded components provide control over the robot functions, as well as access to the sensor data. Higher-level components include the control of the robot movements (based on AI/ML, potentially with a dedicated hardware), the robot configuration, as well as remote access to the robot APIs. The cloud services include basic services for managing map data, status/history data, as well as user and robot configuration data. The robot control component

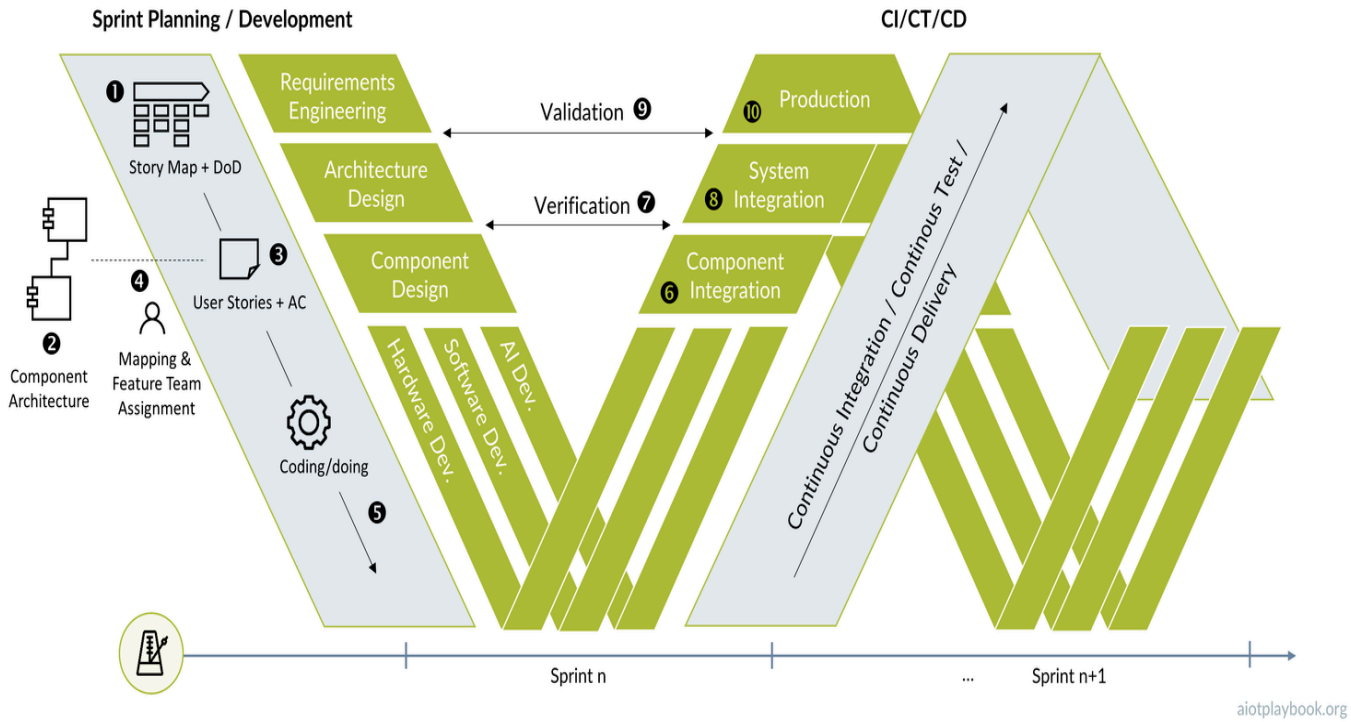
enables remote access to the robot. Finally, the smart phone/mobile app provides components for robot configuration and map management, all accessible via the main screen of the app.



Note that -- in line with the agile "working code over documentation" philosophy -- the component architecture documentation does not need to be very detailed. Only the main functional components are listed here. Depending on the project, usage dependencies can be added if such levels of detail are deemed relevant and the maintenance of the information is realistic.

Sprint n

Next, we want to jump right into the middle of the ACME:Vac development by fast-forwarding to a "Sprint n", which will focus on a single feature to illustrate the approach in general.



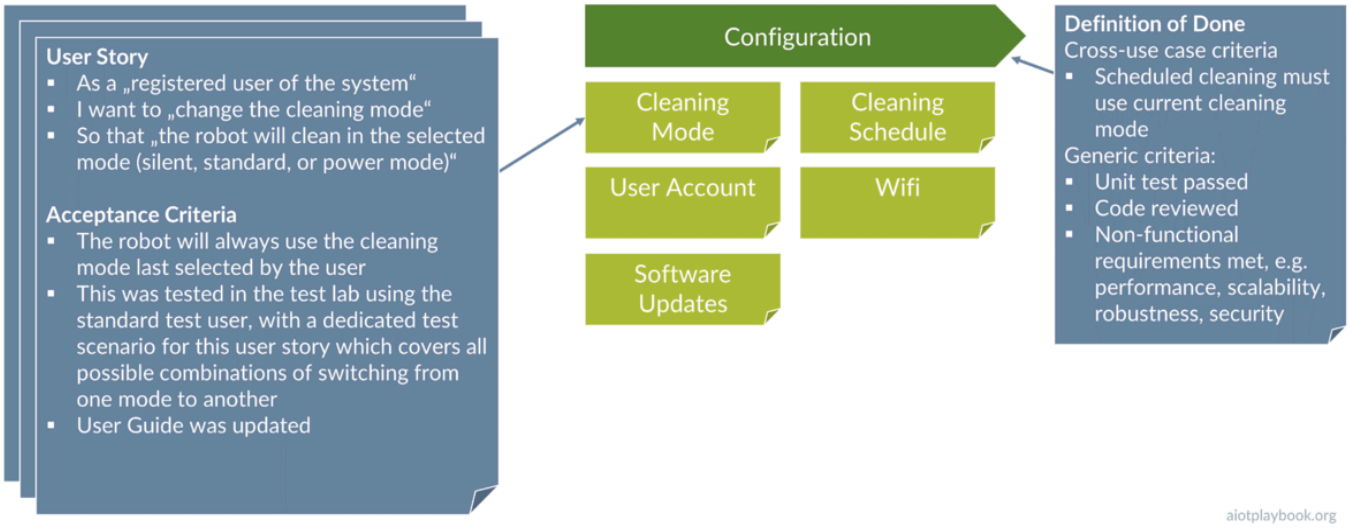
The example will show how a sprint is executed in the agile V-Model, including:

1. Story Map & Definition of Done (DoD): The Story Map -- developed and maintained since "Sprint 0" -- is the topmost agile requirements engineering tool (see Product/Solution Design). This is usually also where the Definition of Done should reside, defining more general acceptance criteria. It is the starting point for our "Sprint n"
2. Component Architecture: Again, developed and maintained since "Sprint 0", this provides an overview of the functional components and their interactions (see AIoT Product Architecture).
3. User Stories + Acceptance Criteria (AC): This is usually the most fine grained requirements definition in the Story Map. User Stories should be concise, fit onto a Post-it Note, and be accompanied with acceptance criteria which are specific to the user story. User stories for this sprint are chosen as part of the sprint planning.
4. Mapping: The next step is to create a mapping between each user story worked for the upcoming sprint and the components which are required in order to implement it. These can be existing components, or components to be newly created. A key result of this mapping is the identification of the required experts to form the feature team responsible for the implementation of the user story.
5. Coding / Doing: In AIoT, the "doing" will not always be coding - it can include hardware development or AI/ML-related work as well. The various tasks at hand will often have different development speeds, and it may not always be possible to create a potentially shippable product increment. The Agile V-Model recommends that if this is a case, at least mockup implementations of the public interfaces should be provided for integration testing.
6. Component Integration: Component integration should focus on integrating and testing the set of components required for a particular user story. This can sometimes mean that these components are embedded into an environment which simulates auxiliary components and/or production data. This should be handled automatically by the CI (Continuous Integration) infrastructure.
7. Verification: Supported by the CT (Continuous Testing) infrastructure, verification will focus on unit tests, as well as testing the compatibility of the components in scope of the particular user story.
8. System Integration: All components changed/created by the current sprint then need to be integrated to create the next, potentially shippable increment of the system.
9. Validation: The validation phase will focus on User Acceptance Tests (UAT). The physical components of the AIoT system should also be tested in the test lab, or undergo field tests. Since field tests can be quite elaborate, they might have dedicated sprints assigned to them.
10. Production: Finally, if all goes well, the sprint results can be moved to the production system. In AIoT, the real production system will most likely only be available after the Start of Production (SOP) of the required hardware. After SOP, taking Edge components into production will probably require use of OTA capabilities.

User Story & Acceptance Criteria

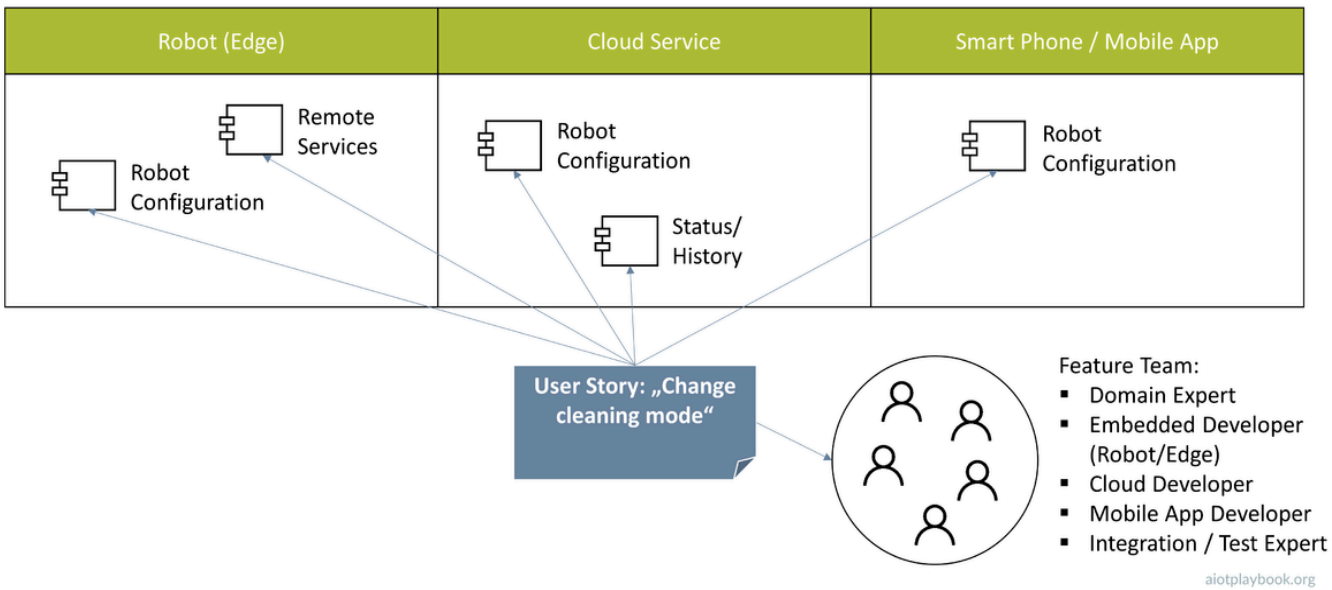
In this example, we are focusing on the ACME:Vac Epic "Configuration". This includes features such as "Cleaning Mode" (e.g. silent, standard, or power mode), "Cleaning Schedule Management", "User Account Management", "WiFi Configuration", as well as "Software Update Management". The Definition of Done provides higher level acceptance criteria.

In our example we focus on the "Cleaning Mode" feature. This contains a Use Story "change cleaning mode", including some acceptance criteria specific to this user story. The intention is that the user can select different cleaning modes via his smart phone app, which will then be supported by both the ad-hoc as well as the scheduled cleaning modes.



Mapping User Story to Components and Feature Team

Having defined the user story, the next step is to identify which components are required for the story. The "Change Cleaning Mode" story will require a robot configuration component on the smartphone. This will need to interact with the robot configuration component in the cloud. In order to record the change, an interaction with the status / history component is required. Finally, the remote service component on the robot will receive the selected mode, and configure the robot accordingly.



Based on the analysis of the functional components, the required members of the feature team for this user story can be identified. They include a domain expert, an embedded developer for the components on the robot, a cloud developer, a mobile app developer, and an integration/test expert. Note that some organizations strive to employ full-stack developers. However, in this case it seems unlikely that a single developer will have all the required skills.

Implementation and CI/CT/CD

Implementation in an AIoT initiative can include many components, including software development, AI/ML development, data management, HW design and engineering, or even manufacturing setup. The various tasks often have to be executed at different development speeds, e.g., because the hardware is usually not evolving as fast as software. Because of this, it might not always be possible to create a potentially shippable product increment. The Agile V-Model recommends that if this is a case, at least mockup implementations of the public interfaces should be provided for integration testing.

Each sprint needs to fully support Continuous Integration, Continuous Testing and Continuous Deployment (CI/CT/CD), with the limitations just discussed in mind. A key element of CI is component integration, which usually should be done with a focus on different user stories, i.e., it should focus on integrating and testing the set of components required for a particular user story. This often means that these components are embedded into an environment which simulates auxiliary components and/or production data. This should be handled automatically by the CI (Continuous Integration) infrastructure.

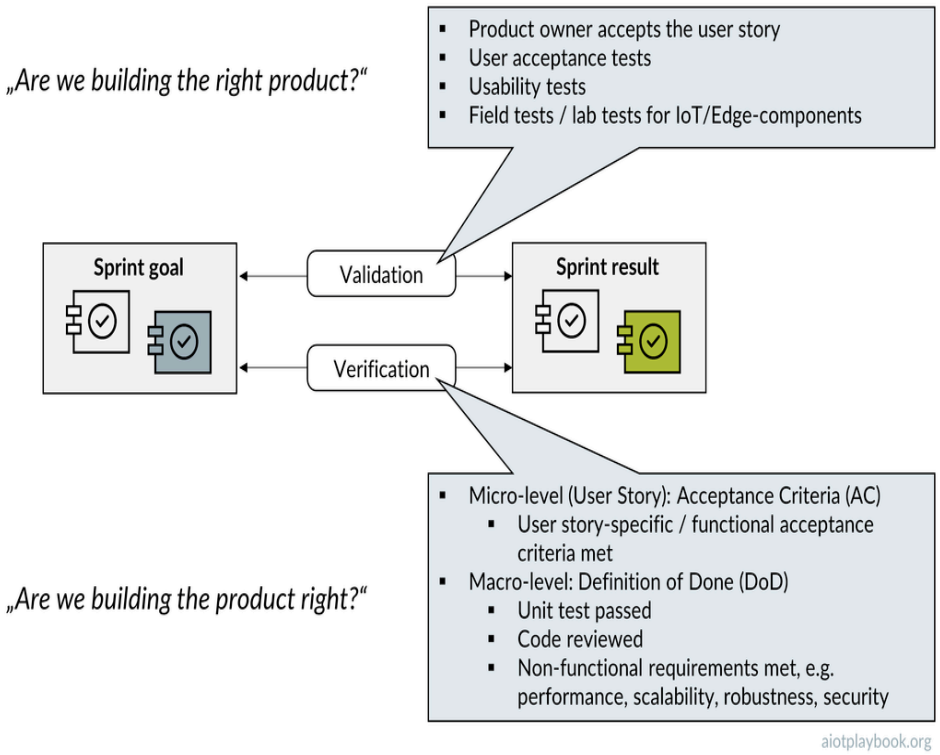
If the component integration including tests was successful, the components can be integrated into the entire system. This means integrating all components which are changed or created by the current sprint need to be integrated in order to create the next, potentially shippable increment of the system.

Verification & Validation

For some people in the agile community, Verification & Validation (V&V) is considered to be an outdated concept, while for many people with an enterprise and/or functional safety background, it is good practice. The Agile V-Model aims for a pragmatic view on this. Componentization (see the Divide & Conquer section) must support an approach where functional components with different levels of QM/functional safety requirements are separated accordingly, so that the most suitable V&V approaches can be applied individually.

Traditionally, validation is supposed to answer the question *"Are we building the right product?"*, while verification focuses on *"Are we building the product right?"*, even though the boundary between these questions can be blurry.

In the Agile V-Model, verification differentiates between the micro and the macro-level. The micro-level is defined by the Acceptance Criteria of the individual user stories, while the macro-level is focusing on the Definition of Done, which usually applies across individual user stories. Validation, on the other hand, focuses on user acceptance. In AIoT, this can involve quite laborious lab and field tests.



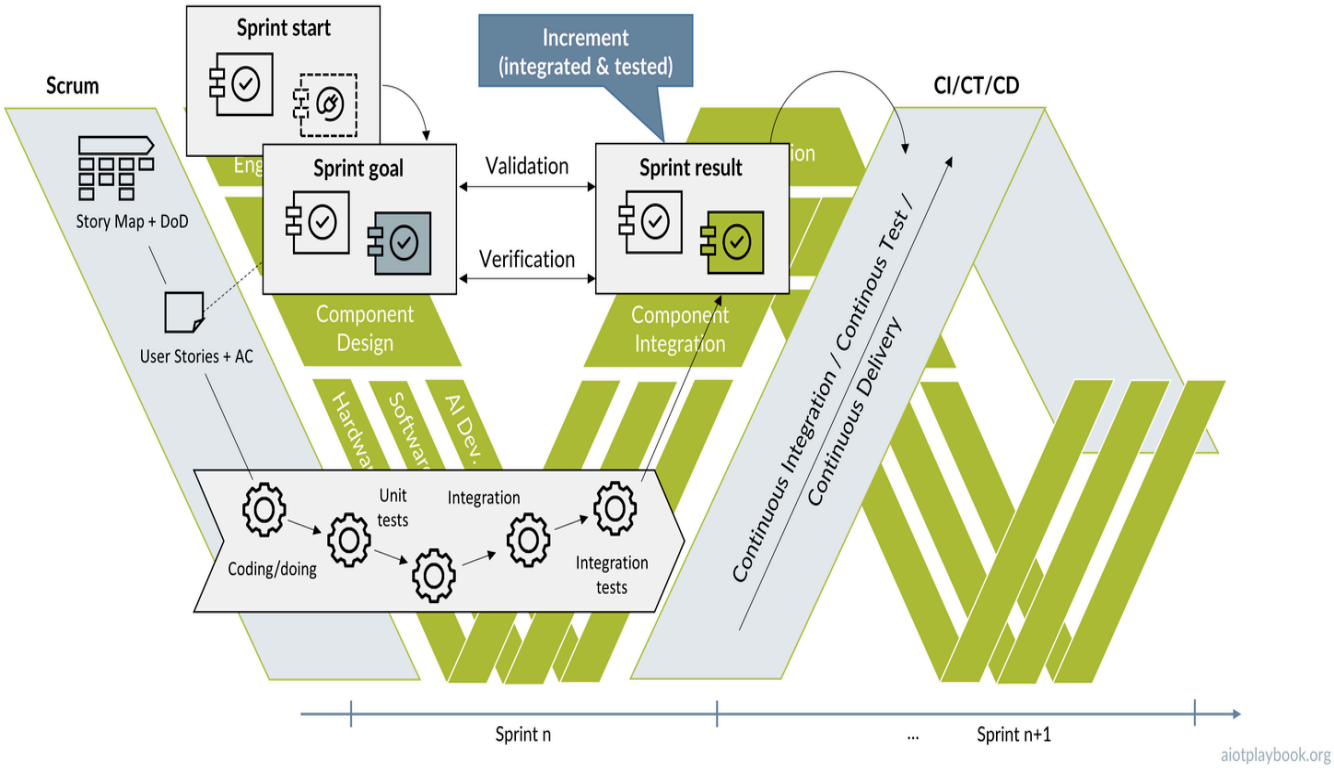
Sprint n+1

At the end of a sprint, the sprint retrospective should summarize any key findings from the sprint's V&V tasks that need to be carried over to the next sprint. For example, findings during User Acceptance Tests might require a change to a user story. Or a feature team might have decided on an ad hoc change to the planned component architecture. These kinds of findings must either be reflected by changes to the backlog (for user story definitions), or by updating the architecture plans and documentation.

Summary

In summary, the Agile V-Model is designed to support the adoption of agile practices in an AIoT environment, where we usually find some significant inhibitors of a "pure" agile approach. Each v-sprint combines a normal, agile sprint with a sprint-specific planning element. This allows us to address the complexities inherent to most AIoT initiatives, as well as the typical V&V requirements often found in AIoT.

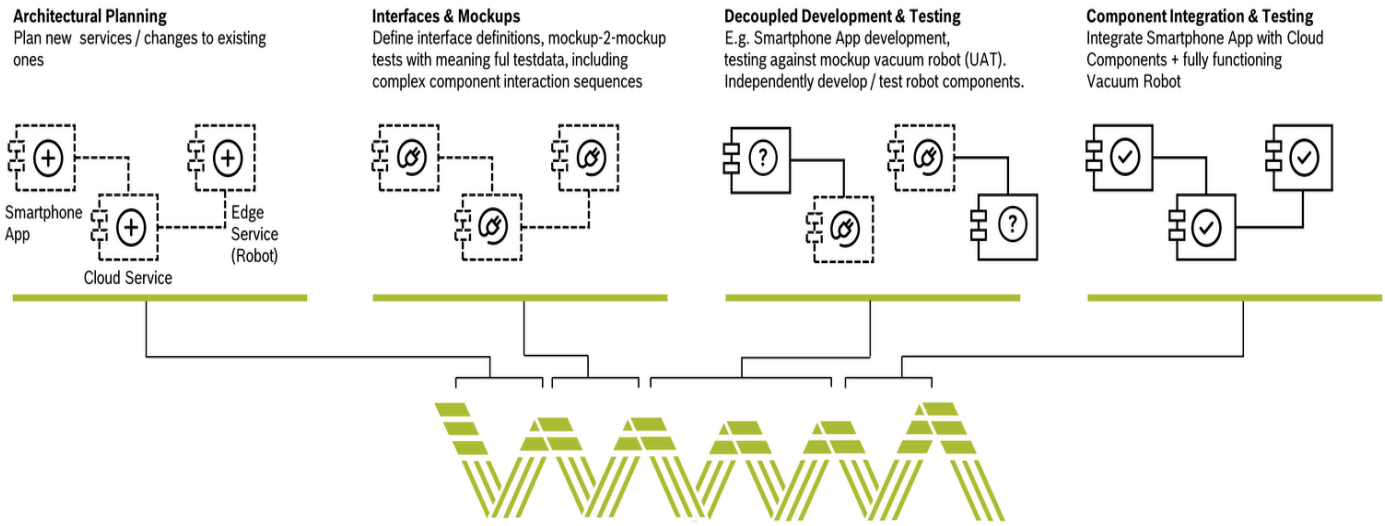
At the end of each sprint, the sprint results must be compared to the original sprint goals. Story maps and component landscapes must be adapted according to the results of this process. The updated version then serves as input to the planning of the next sprint.



Decoupling Development

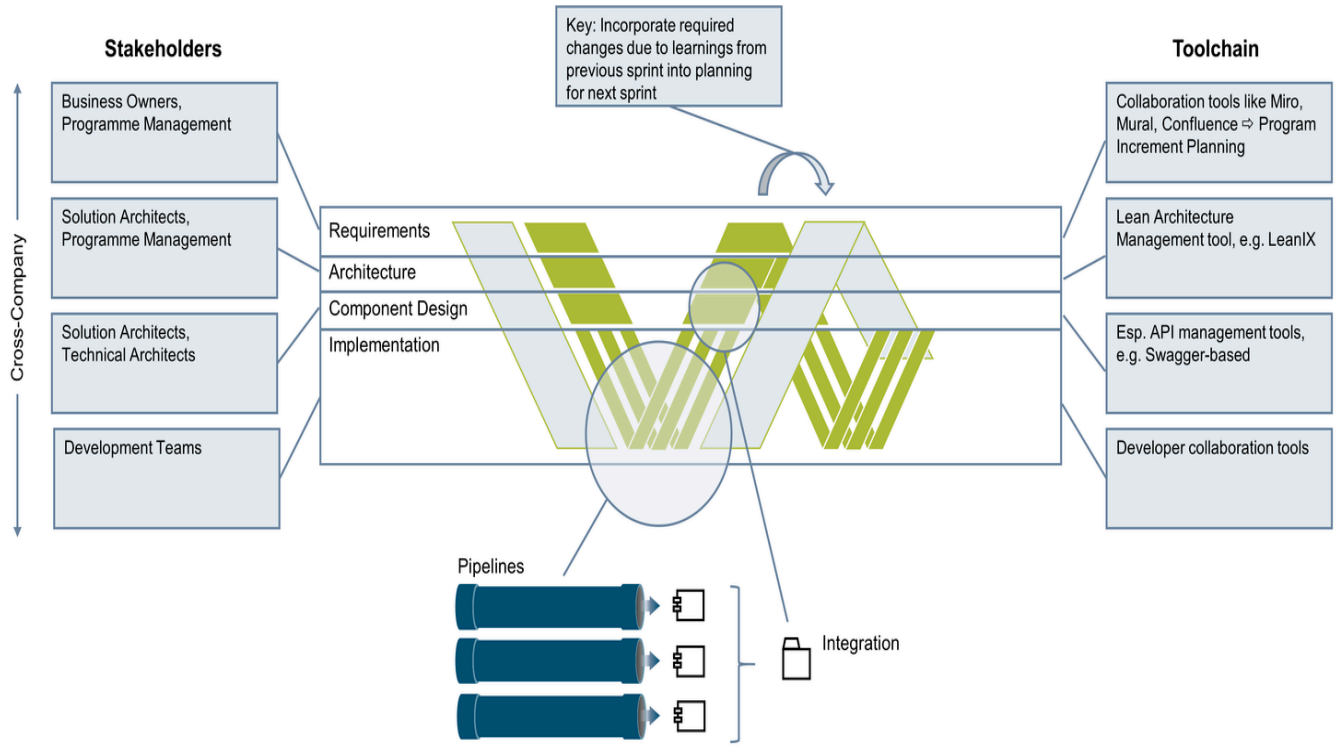
To master the complexity of an AIoT product or solution, it is important to apply an effective "Divide & Conquer" strategy. This requires decoupling both on the development and on the technical level. In order to decouple the development, an interface-first strategy must be implemented. This will allow for the development and testing of components independently of each other, until they have reached a level of maturity so that they can be fully integrated and tested.

This will also be especially important in AIoT because it will not always be possible to test all components in the context of the full system. This is why many AIoT developments utilize different test simulation approaches. These can range from simple interface simulations (e.g., via REST tools) to sophisticated virtual simulations (e.g., allowing us to test an algorithm for autonomous driving in a virtual world). Since field tests can be time-consuming and expensive, this is important in AIoT.



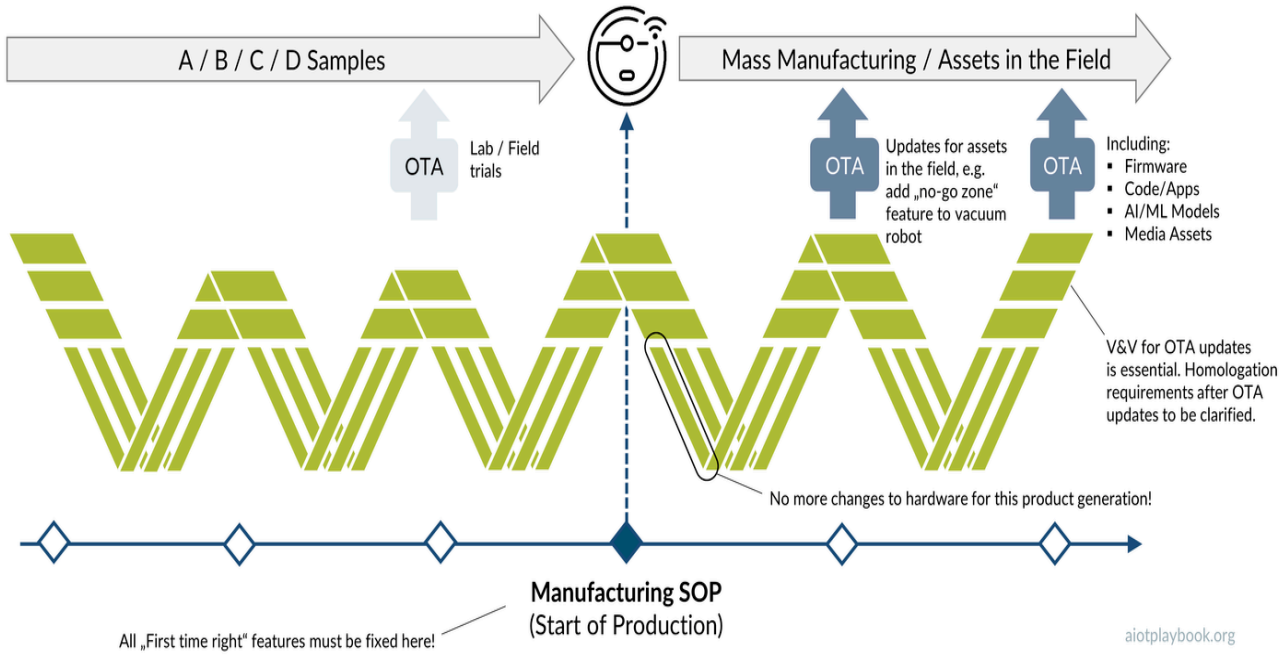
Stakeholders and Collaboration

The Agile V-Model aims to support collaboration between different stakeholders who are involved in the execution. Very often, this will involve cross-company collaboration, e.g., if certain parts of the AIoT system are developed by different companies. It is important to think about which tools should be used for efficient stakeholder collaboration, both within individual stakeholder groups and between them. The figure following provides some examples. Finally, on the technical level it is important to note that the Agile V-Model must support the different AIoT pipelines, as introduced earlier.



Agile V-Model and AIoT

Finally, this discussion should conclude by mentioning what is probably the biggest difference between a "normal" software project and an AIoT project: the Start of Production, or SOP. This is the point in time when mass production of the smart, connected products is starting, and they are leaving the factory to be deployed in the field. Alternatively, the start of the roll-out of retrofitting the AIoT solution to existing assets in the field. Any required changes to the hardware configuration of the product or solution will now be extremely difficult to achieve (usually involving costly product recalls, which nobody wants). In a world where manufacturers want to utilize the AIoT to constantly stay in contact with their products after they have reached the customer, and provide the customer with new digital services and applications -- rolled out via OTA -- it becomes extremely important to understand that the V&V process does not stop with the SOP, at least not for anything that is software. With software -defined vehicles, software-defined robots, and software-defined everything, this is quickly becoming the new normal.



Issues and Concerns

The proposed Agile V-Model approach has provoked intense discussions between advocates of the different worlds. Some of the arguments against the approach have been captured in the table below. Some comments and counterarguments are included as well. We hope that the discussion will continue, and we will be able to capture more pros and cons.

Issue / Warning about the V-Model approach	Remarks
Don`t fall back into RUP-style, super detailed documentation which is costly and always outdated.	Very true, this must definitely be avoided. The examples provided here are hopefull showing how this can be avoided by keeping it on the right level of detail and combining it with agile best practices. Still, in complex projects a certain level of high-level architecture documentation will be required to align all stakeholders.
This is introducing too much processual overhead.	Maybe true. Depends on how you set up the Agile V-Model in your organization. Plus, some organizations still feel more comfortable with at least a certain level of process.
You should not centralize requirements management and architecture decisions. The outcome should be driven by the Definition of Done.	Partly true. The Agile V-Model is proposing to combine Definition of Done (per feature/use case) or Acceptance Criteria (per User Story) with a lightweight, overarching system design to ensure cross-team alignment.
Sprint planning must be done before the next sprint starts.	True. The planning of the next sprint must start while you move up the right side of the previous V.
Strict decomponentization too early in the development cycle can cause problems further downstream, if not done right.	True. Finding the right structure for the component landscape is key, and often requires multiple iterations. This is a strong point of the Agile V-Model, because the high-level component landscape is revisited at the beginning of each v-sprint.
V-Model has a bad reputation, the name is tainted	Maybe true. The name was chosen to provoke a discussion. Let`s see where it goes and whether we have to reconsider this.

aiotplaybook.org

Expert Opinion

Sebastian Helbeck is a VP and Platform Owner Power Tools Drive Train at Bosch Power Tools. The following his summarizes his thoughts on this topic.

Dirk Slama: *Do you see the agile world (cloud, etc.) and the waterfall world (e.g., embedded) currently working well together? What are the issues?*

Sebastian Helbeck: *What I am seeing is that many projects are currently developed parallel in these two worlds. This is based on the fact that in many cases either the embedded or the non-embedded part already exists, and the other part needs to be developed. In the future this needs to be done more seamlessly.*

DS: *How much do these worlds need about each other, and how can we ensure this?*

SH: *Currently everybody is focusing on understanding and fulfilling their own contributions and the interfaces around. In the future, the interfaces will change and will be even more complex. Edge computing will be a key driver here. Consequently, we need to have better knowledge about our own contributions and how they relate to the other side. As an analogy, FPGA technology can be used, which has tremendously changed the interfaces between HW and embedded SW.*

DS: *Can the different methods coexist, or do we need a common approach?*

SH: *Currently, they coexist and it works, but it is not ideal. This is why as the next step, we need to bring the different approaches closer together due to the different maturities of the product parts. To unleash the full power of AIoT, we need to rethink this and have to adopt a more holistic approach.*

DS: *Which role can the Agile V-Model play here?*

SH: *It can potentially play an important role here. It will combine the two worlds with a holistic view to give all the included stakeholders a transparent overview of the complete system. For the future, we might need to think about a new name since the combination of the worlds will bring us to a new dimension of understanding of our AIoT systems.*