HARDWARE & SOFTWARE AGILE TECHNIQUES

WHITE PAPER-

This NDIA supplementary white paper provides Agile techniques for hardware and software development including team member cross-functionality, crafting vertical slices of product, developing products incrementally, and Scaling.

Denise Jarvie & Jessica Crowley
H&A Radically Better Agile

HUMPHREYS & ASSOCIATES
Radically Better Agile

Contents

HUMPHREYS & ASSOCIATES
Radically Better Agile

Agile Techniques for Hardware & Software Development

Introduction

This white paper presents best practices and techniques to overcome hardware and software development challenges encountered as organizations move towards increased agility. Agile transformations typically start in pockets within the same discipline, such as Software or IT departments. As the Agile teams become more efficient and require increased support from inter-related departments, the supporting functions will likely adapt and begin to use Agile techniques to keep up with increased demand. Expanding Agile across the value stream results in the need for the organization to assess their current state of agility, create cross-functional teams, decide how to scale, craft vertical slices of work, adapt new development techniques, and increase customer involvement. Teams will need to be restructured to allow for end-to-end development, which is particularly difficult for hardware teams.

Team Structure & Composition

Hardware team structures are typically multi-disciplined in very diverse areas of competence. The Agile Coach should analyze the team composition and ensure all skills necessary to fulfill the team vision are present. It is possible for a hardware team to be end-to-end, with the use of techniques such as deployment of people with scarce skills, support from Subject Matter Experts (SMEs) and functional roles at the scaled level to ensure that the team can be self-reliant. If a skill set gap is present the coach can facilitate training, skill pairing, team deployment, or hiring so that the team can independently produce working product.

The first step towards cross-functionality on an Agile team is to pair similar skills together on as many stories as possible. An Electrical Engineer and a Mechanical Engineer would be an example of a natural pairing as their education and working experience have many parallels. As time goes on the team members will break down development barriers that typically exist between departments. This encourages the team members to become more T-shaped, which means deep in one area of competence and knowledgeable in other areas. They will understand the impact of “throwing work over the wall”. They will create more efficient ways to do work. They will need access to tools and assets within their own discipline that have been restricted to one or two individuals. An example of where restricted access may occur is within the Mechanical Engineering discipline. The development team may be separate from the Test Lab team, meaning they share the same discipline but only the Test Lab team has access and training on test equipment.

Software team structures are typically rooted in the same discipline; however, they are segregated by stage of software development and access to tools. For example, you may have a software team that is composed of an embedded software engineer, a software development engineer, and an application

developer. Some team members may have experience in legacy products and access to specific tools while newer team members may be focused on new product development. It is much easier to break down inter-departmental barriers than it is to break down cross-departmental barriers.

Crafting Vertical Slices of Product Functionality

Product Roadmap development forms the foundation for Agile product architecture that allows for carefully crafted user stories and slices of work so that multiple disciplines can work on the same scope. The first layer of the roadmap should consist of Epics which are composed of large pieces of scope. Epics can be developed in a modular, phased, or business / user experience approach to help establish initial product architecture that enables teams to avoid regressing to waterfall development techniques.

For new programs, customers can support Agile development approaches by using a Work Breakdown Structure (WBS) that is product based, focuses on outcomes, and is compliant with Mil-STD-881. Customers should play a major role in Product Roadmap Development and ensure that Epics are architected to support vertical slicing and are not sequential/departmental pieces of work. The requirements for technical reviews should be modified to allow for incremental and iterative reviews and will require more frequent participation by the customer.

For existing programs that have recently transitioned to Agile development techniques, the customer should request Agile–Earned Value hierarchy and mapping. The Product Backlog must cover all requirements for the work that is being executed by development teams. The requirements are satisfied by the Stories, Features, and Epics. They should map to the Statement of Work paragraphs and Work Breakdown Structure. The customer should make time for more frequent Product Owner interaction and may need to access the Agile tools to have a better understanding of the Product Backlog and Release Plan. Attending the Release Backlog Reviews and major Release demonstrations will provide the Agile teams with crucial feedback that is necessary for continuous improvement.

Before the Integrated Baseline Review (IBR), the customer should have an in-depth understanding of the Agile techniques, frameworks, terminology, hierarchy, mapping and product architecture that is planned to be used on the program. This will allow the IBR discussion to be focused on how realistic and achievable the baseline is instead of discussing the chosen Agile approach.

Waterfall Epics consist of departmental or sequential pieces of scope. It would be difficult to design, develop, test, integrate and produce incremental pieces of working product using this type of product architecture. Teams would have to complete all Epics to arrive at working product.

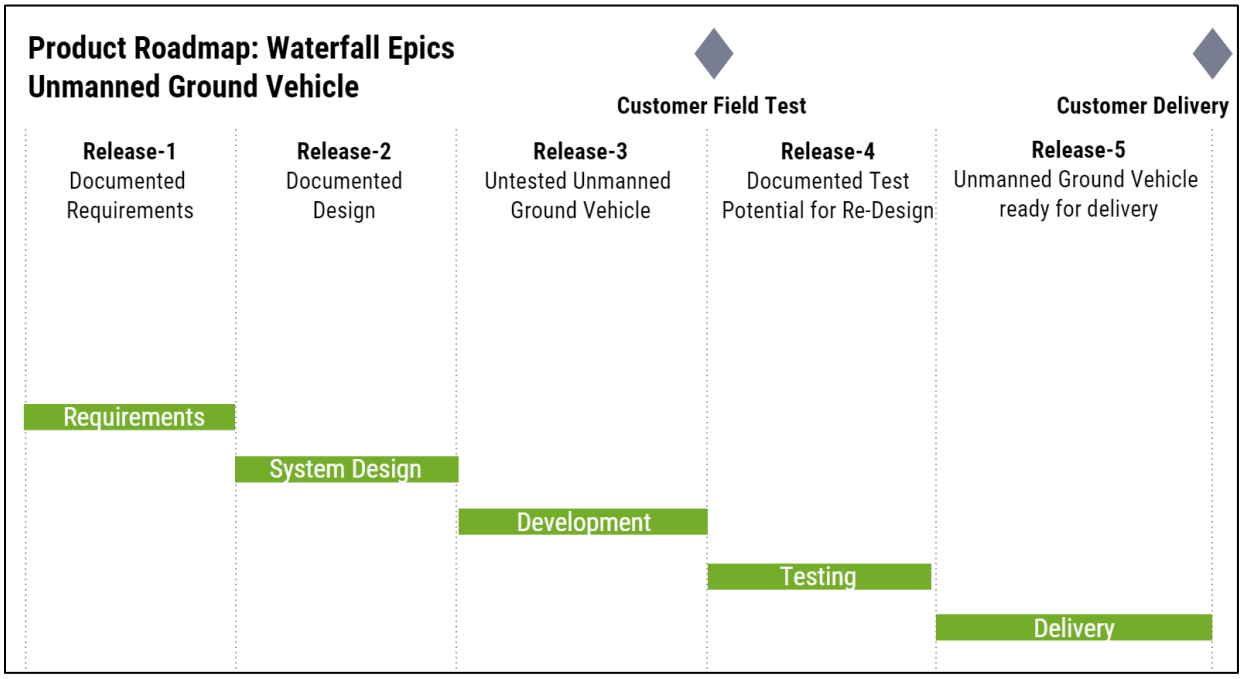

Figure 1- Waterfall Epics

In the above diagram, the team may be cross-functional and fully able to develop the product end-to-end but the way the work has been architected encourages the team to work within their departmental functions in a sequential fashion.

Epics can be architected into “modules” or parts of a product. Each module or part needs a Known Stable Interface (KSI) between itself and other modules or parts. This helps ensure the integrity of the relationships between them. The Product Ownership team serves as the KSI between modules and handles inter-dependencies and system integration at a scaled level. Modules will need to conform to the constraints surrounding the entire system or product. An example of this would be envelope constraints. If a module needs to fit into an existing product it would have to conform to a specific envelope constraint; otherwise it would not fit.

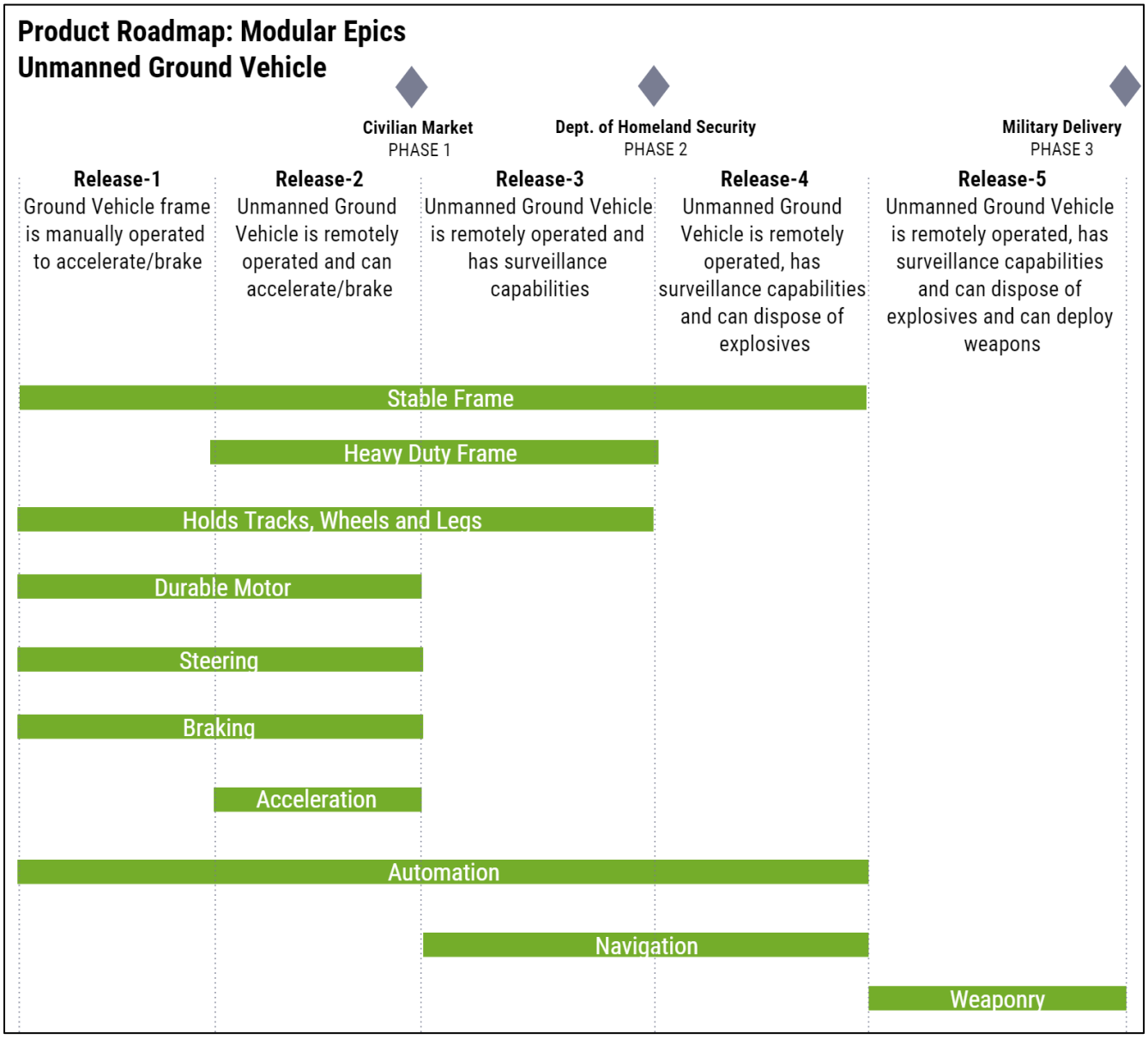

Figure 2- Modular Epics

In the above diagram, the Epics have been architected into individual parts or modules, so there must be significant focus on integration and system level thinking to avoid extensive re-work. Breaking Epics into modules is one of the easiest approaches that a Product Owner can use to group work because it allows them to visualize and manage the pieces of scope and interfaces in a way that can be easily understood. The weakness in this approach is that the roadmap is not as customer centric as other techniques.

Epics can be architected into waves or "phases" of capability. This is a technique that is best suited for a product that is delivered to a customer in various states or is delivered to multiple phases of customers. The first phase is typically a minimum viable product (MVP). Additional phases build on the MVP to refine and improve the product allowing it to be released to other customers or end users for different purposes. This continues until the product meets the primary end user or customer's needs.

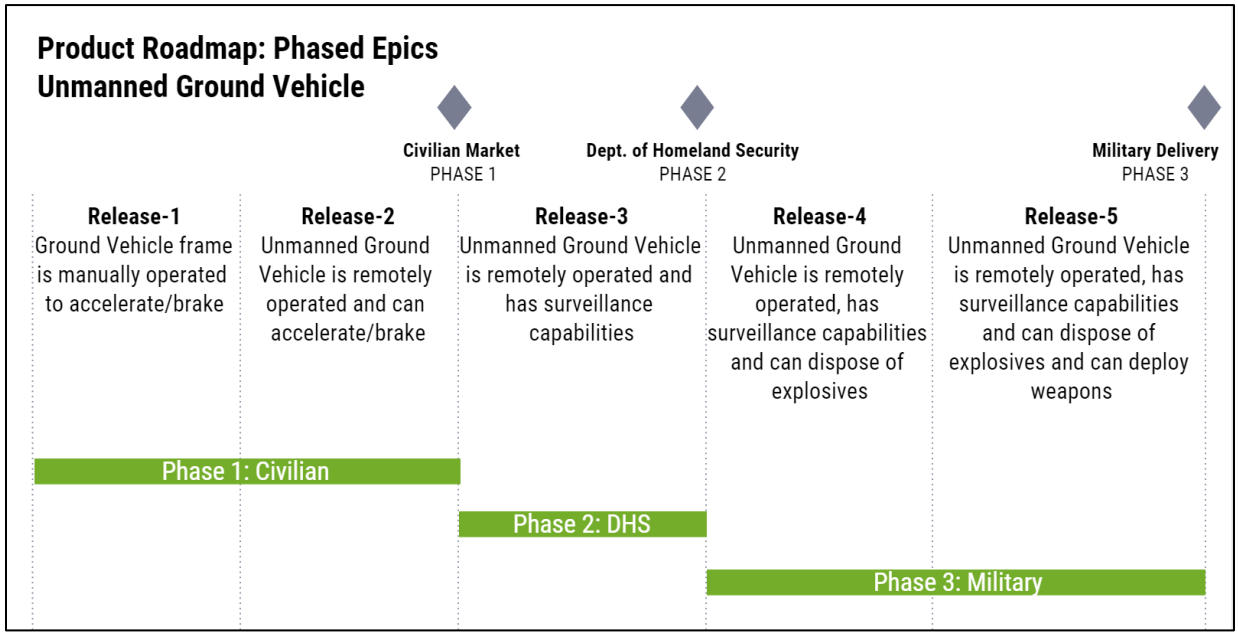

Figure 3- Phased Epics

Epics can also be architected as a part of the existing business flow or desired user experience. Epics are organized around the steps the business or customer currently takes, or desires to take, to create or use a product. Epics formed in this manner will draw the organization's focus to the customer's desired outcome.

Figure 4- Customer Personas

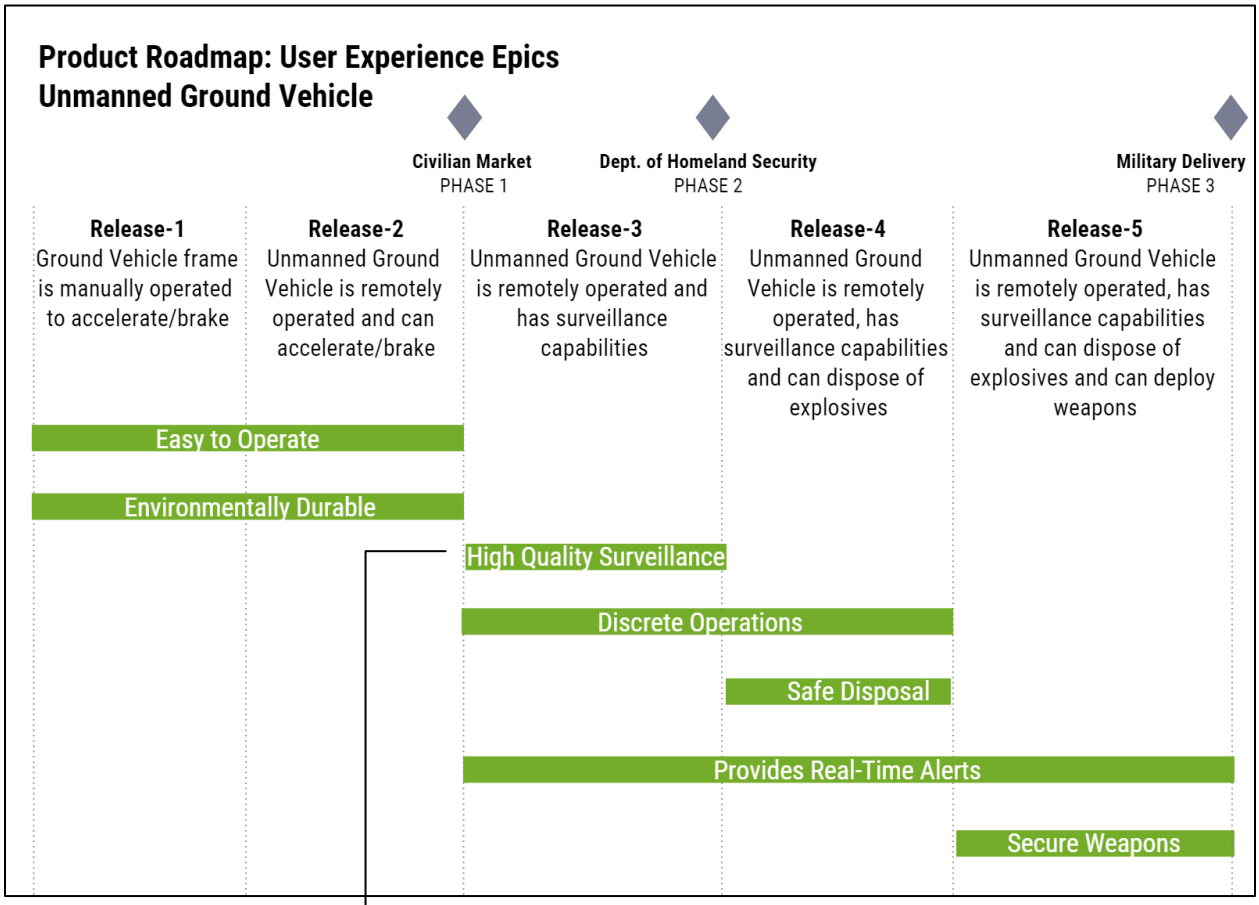

Figure 5- User Experience Epics

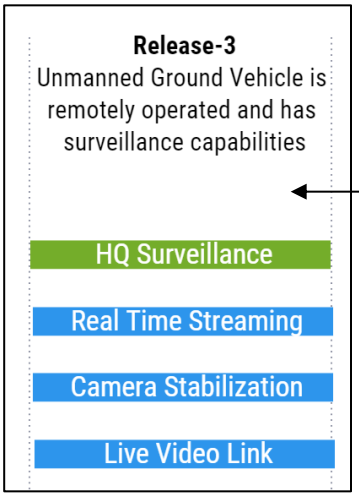

Figure 6- Feature

Regardless of the chosen method for Epic development, each Epic can be further decomposed into Features that represent a slice of meaningful functionality to a customer or end user. When the teams are ready to work on a Feature they will break it in to user stories.

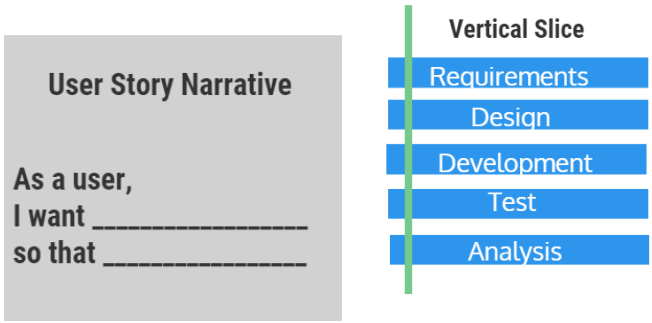

Figure 7- User Story Formation

User stories are typically written in narrative form and should be a testable vertical slice of functionality. The story should encompass requirements, design, development, test, integration and analysis.

Each user story should have a Definition of Done and an Acceptance Criteria. The Definition of Done is a checklist for user stories of similar types and ensures quality and completeness. The Acceptance Criteria helps the team have a clear understanding of the problem they are trying to solve and what test they will have to pass before work will be considered complete. The Acceptance Criteria clarifies the "what" and ensures there is a common understanding of the requirements prior to the start of work.

Vertical slices allow the team to quickly pivot as they learn more about the product they are developing. Because the product is being incrementally tested and vetted by the customer, there is a higher probability that the customer will be happier with the end product and that the product itself will be higher quality. Once vertical slices are ready to be worked by the team, there are many techniques they can use to perform the work with increasing agility.

Hardware Development Challenges

Hardware development is physical in nature therefore it is significantly different from the intangible development of software products. Changes to hardware are generally more expensive due to wasted development efforts, raw material costs, shipping and handling, expediting fees, inventory, and material management costs. Drastic software changes are virtual in nature and so they typically result in re-work and wasted development efforts. This significant difference in the cost of altering a design makes it imperative that hardware teams have less expensive ways to test out their ideas early in the product development process, before ordering more expensive prototype, pilot, or production materials. This is one reason why teams brainstorm many design concepts for components and use virtual modeling, 3-D printing, and simulations to fail the concepts as early as possible, obtaining meaningful customer feedback along the way. They then arrive at the best design solutions and continue to use virtual modeling, 3-D printing, and simulations during sourcing lead times to gain important product feedback about the slice of functionality they are developing. The teams push to build "something" demonstratable to the customer to solicit feedback regardless of prototype, pilot, or production part availability.

Hardware development introduces a special challenge in the form of lead time associated with obtaining a supply of raw materials. As the team rapidly iterates toward developed products, they need a shorter

supply chain with vetted and available rapid sourcing suppliers. Adjusting the supply chain to accommodate quick turn suppliers and rapid prototyping takes time. An initial workaround for shortening the onboarding time of prototype suppliers in the Enterprise Resource Planning (ERP) system is the use of purchase cards available to the development team with an associated spend limit. This allows the teams to purchase low cost prototype materials from the supplier of their choice without the red tape associated with getting interim suppliers in the ERP system. A more lasting solution is the development of the sourcing skillset on development teams or the inclusion of sourcing at a scaled level. This will allow the teams to either source materials themselves or access more timely sourcing at the team of teams.

Agile Development Techniques

Many of the agile best practices and techniques that software teams use can be modified and applied to hardware development to create more opportunities for agility along the value stream.

Pair Programming is an agile software development technique in which two programmers work together at one workstation. One person writes code (Driver) while the other reviews each line of code as it is typed (Observer/Navigator) and strategizes about future work or improvement opportunities. The two are both capable of writing code and regularly switch roles. Because the two software team members pair together, bugs are less prevalent and the two are more accountable to efficiently complete work. Many studies have been conducted on the effectiveness of Pair Programming and the results show that pairs have ~15% less defects in their code. However, pairing can become overused if team members spend all day working together and are unable to independently approach work. It can also be distracting to use pair programming for more complex code in which solo work is better suited.

The concept of “pairing” is derived from the Pair Programming technique and can be used to facilitate Development Team Members working together to complete stories for the purpose of shared learning, awareness of the “big” picture, collaboration, and increased knowledge saturation. Pairing is an easy way to start making gains towards a more cross-functional team.

Test Driven Development is a software development technique, also known as red-green refactoring, that requires developers to write and run tests before they write code. The new test is run and subsequently failed to ensure that it captures the necessary logic to be confident in the results. The minimum code is now written to pass the test and the test is repeated to ensure the code passes. If the code fails it will be adjusted until it passes. The code is re-factored, non-functional attributes of the code are improved, and the result is solid code, reduced re-work, and improved quality.

Agile hardware teams borrow from this concept by writing stories with a test in mind and using red-green testing techniques to develop the product to the point that it passes the test. This is often referred to as the Minimum Viable Product (MVP) and it helps prevent over-engineering of the solution.

This approach also ensures that work is completely done each sprint and it raises the quality of the final product because the pieces are tested along the way.

Scaling and Infrastructure

Agile development techniques can improve the efficiency within teams; however, there may be dependencies between teams, departments, or supporting functions that need to be resolved in order to arrive at complex, large scale product solutions. Scaling can help address bottlenecks and significantly improve time to market as waste is removed from current development processes. Some examples of organizational waste include handoffs, specializations, multi-tasking, context switching, and decision making by parties not involved in the details of development.

Agile Heat Mapping is a technique where the location and maturation of Agile, lean techniques, frameworks and best practices are mapped out and recorded across the organization. This provides visibility into pockets of Agility and allows for enhanced collaboration, learning, and sharing. It also serves as a baseline for Horizontal Scaling. Horizontal Scaling refers to the spread of Agile approaches across the organization. Teams that are using Scrum, Kanban, various lean and agile techniques, and scaled frameworks can be present within the same organization and yet they are oblivious to each other's existence. By mapping out the current state of organizational agility, progress can be tracked over time.

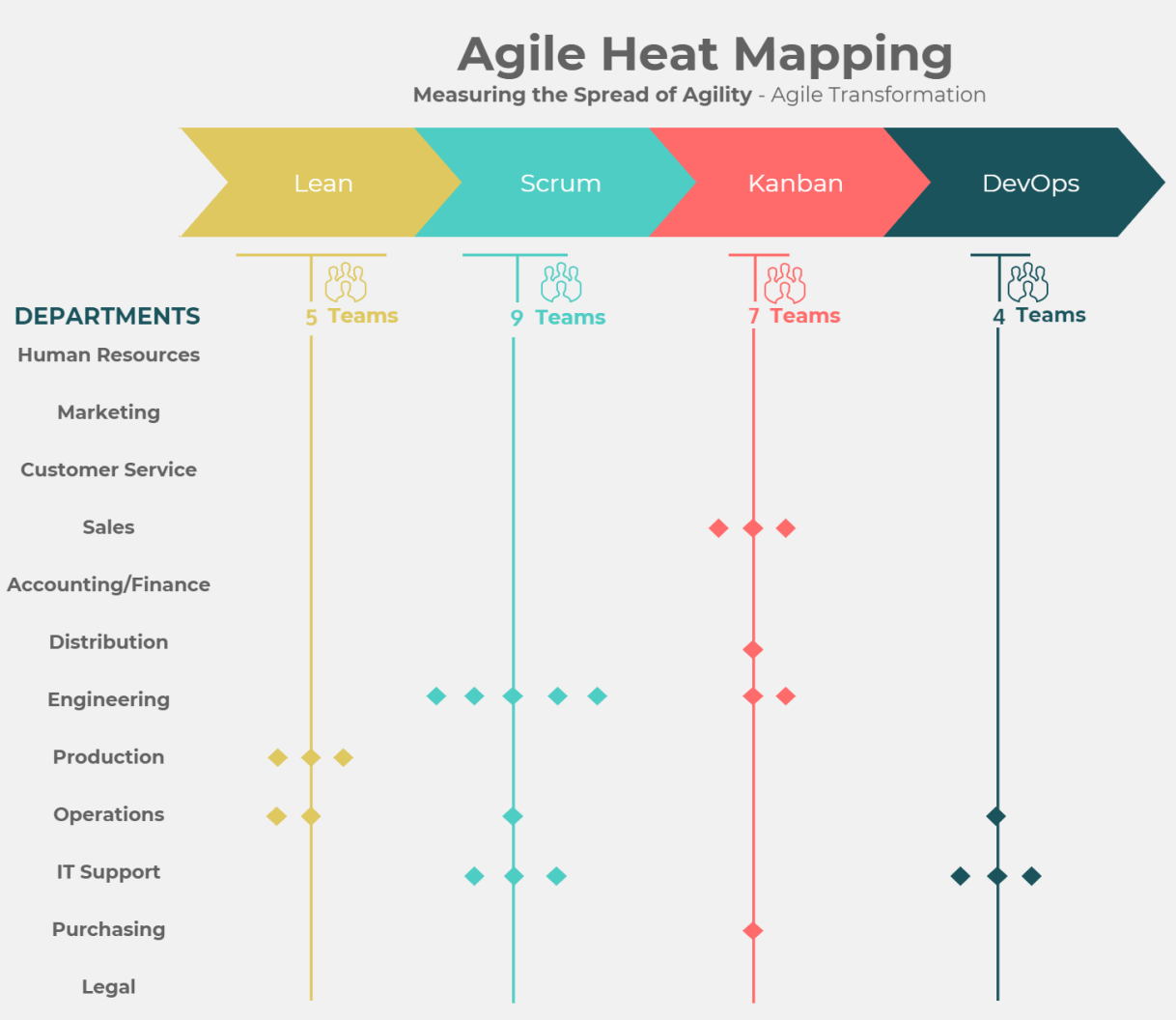

Figure 8- Agile Heat Mapping

As Agile pockets are discovered and teams push for greater increase in agility, they need to combine efforts in order to efficiently utilize resources and eliminate waste. Value Stream Mapping is an activity where the flow of work through the existing structure is laid out and sequenced. It is all-encompassing from product inception to product production. It helps make inefficiencies, waste, and bottlenecks visible and highlights improvement opportunities that can lean out the value stream. Value stream mapping can provide insight into organizational areas that would benefit from additional Agile infrastructure and scaling. Vertical Scaling establishes infrastructure in the form of an Agile Framework to support development teams as they work. Vertical scaling provides alignment and structure to manage dependencies, resolve impediments, prioritize work and align teams along the value streams. Without this type of infrastructure, the teams are less efficient at system level integration, have more hand-offs, and do not operate as a whole unit with the big picture in mind.

Conclusion

In conclusion, there are many best practices and techniques that organizations can use to overcome hardware and software development challenges during an Agile transformation. The migration from departmentalization to cross-functionality can be accelerated using techniques such as value stream mapping, skill gap analysis, pairing, and the deployment of scarce skills at the scaled level.

Modifying Agile software development techniques for use in hardware can improve the quality of product and reduce time to market. The difference between software development and hardware development is intangible versus physical product, which can result in costly incremental design changes. Access to virtual modeling, 3D printing, and simulations allows teams to iterate and minimize hardware development expenses.

Customers and contractors can work together to develop the product roadmap and architect work in a way that enables teams to develop vertical slices of functionality. The inclusion of definition of done and acceptance criteria, and the use of user story narratives ensures that the work is completed with higher quality and fulfils all requirements.

Some of the greatest challenges in Agile Transformation surround team composition, product architecture, stable infrastructure, and customer involvement.