Embedded Systems Architecture for Agile Development

A Layers-Based Model

Mohsen Mirtalebi

Apress®

Embedded Systems Architecture for Agile Development

A Layers-Based Model

Mohsen Mirtalebi

Apress®

Embedded Systems Architecture for Agile Development: A Layers-Based Model

Mohsen Mirtalebi
Indianapolis, Indiana, USA

ISBN-13 (pbk): 978-1-4842-3050-3 ISBN-13 (electronic): 978-1-4842-3051-0
https://doi.org/10.1007/978-1-4842-3051-0

Library of Congress Control Number: 2017957715

Cover image by Freepik (www.freepik.com)

Managing Director: Welmoed Spahr
Editorial Director: Todd Green
Acquisitions Editor: Susan McDermott
Development Editor: Laura Berendson
Technical Reviewer: Amir Shahirinia
Coordinating Editor: Rita Fernando
Copy Editor: Karen Jameson

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the books product page, located at www.apress.com/9781484230503. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

To my parents who put up with my handyman projects since age four when I was first electrocuted attempting to fix the TV. To my son who has patiently learned that I only answer the calls after the sixth attempt. This is why I was slow.

Table of Contents

About the Author

Mohsen Mirtalebi is a specialist in diagnostics software with Cummins Inc., a global power leader that designs, manufactures, sells, and services diesel and alternative fuel engines as well as related components and technologies. He has more than 10 years of experience in the engineering field. At Cummins, Mohsen leads a team of engineers, where he oversees the software quality meeting regulatory requirements by utilizing various data analysis tools, controls software reviews, as well as Agile and project management tools. Previously he worked for Rockwell Automation as a control firmware engineer, responsible for control algorithm design and implementation for motor drives utilizing MBD, and real-time operating systems. He has held other hardware/software positions at Danfoss, Emerson Process Management, and more. Computer skills include being a certified user of Matlab/Simulink, LabView/TestStand and Texas Instrument DSC/DSP products specializing in motion devices. Mohsen has a MS in Electrical Engineering with an emphasis on power electronics control and HIL/SIL/MIL. He has been a member of IEEE for 10 years. He is an advocate of STEM and has coached many robotics teams in grade schools.

About the Technical Reviewer

Dr. Amir Shahirinia is an Assistant Professor of the Department of Electrical and Computer Engineering at the University of the District of Columbia, Washington, DC. He received BS and MS degrees from K.N.Toosi University of Techology, Tehran, Iran, and a PhD from University of Wisconsin-Milwaukee in Electrical Engineering. He also performed postdoctoral research in the Power Electronics group at Rockwell Automation (Allen Bradly) from 2013–2015. Dr. Shahirinia is the director of Center of Excellence for Renewable Energy (CERE) at UDC. Dr. Shahirinia's research interests encompass the areas of power systems, smart grids, power electronics, and control and ranges from optimal planning of renewable energy grid integration systems (REGIS), optimal operations of REGIS, modeling and intelligent real-time control of REGIS, Bayesian statistical analysis and predictive modeling of REGIS, to power electronics and motor drives.

Introduction

What do onions and software have in common? Apparently there is nothing in common between them, but how many things around us are inspired by nature? Years ago I was hired by a high-tech company for what was, I thought at the time, one of the coolest products. Here I was in a small manufacturing company with a high innovative spirit but not part of a highly meaningful organization. For a duration of two and half months I had nothing to do but to get myself familiarized with the product, and I loved it. So as a free agent I started roaming around every corner of the company, visiting many departments from quality control to hardware, manufacturing, program management, and even product returns.

The result of two and half months of my spiritual journey into the deep belly of the product was to discover that there were numerous holes in our development process of which many were software related. Since our product target and host software were closely coupled, which were also utilized internally for various applications such as product calibration and tests, I came up with a 30-page report including some recommendations to improve the quality of the software architecture on both ends: target and host software. I thought it could be cost effective to enhance the existing architecture rather than investing in a new platform. Additionally, there were some serious security concerns. Our OEM customers were using the same host software, and the chances of breaching the engineering tier was high. Perhaps when our OEM customer was talking to us, we didn't do a good job of listening; and now they had to take the matter into their own capable hands to tweak some product configuration parameters for us.

On that account, listening to the customer's voice is essential. Either we choose to listen to them when we are developing the requirements or were forced to do so when the hammer comes down on us through the product return doors. Although we might think the product research and returns are two different departments, in fact they are the same with a minor difference. These two departments fall on the two ends of product development process, but in both, people get into the same type of cause-and-effect cycles. The difference falls into the chronology of the product issues.

A new anomaly in returns most likely is an old known one in research. The reason some people might think it is a new problem is because the problem was either overlooked, forgotten, or currently being worked on – in secrets without notifying other internal departments. So if there was a way we could capture this wealth of knowledge that we've tirelessly gained during product R&D, then there is a very good chance we can better manage them downstream. This is called transparency.

In another example with a different manufacturer, I observed that the products of a well-known test and measurement vendor were extensively used in the R&D department. Although the manufacturing was not using the same type of test tools, but their test routines were basically the same only designed to be more subjective and shorter in duration. Then I thought if I could convert the tools in manufacturing to be matched with R&D tools, I could recycle research software programs for manufacturing uses. So that was exactly what I did. The result was a seamless path between development and manufacturing that not only unified the tools but also the languages these two very different departments were speaking. This was more than creating a scalable test process; it was about creating a freeway of knowledge – the very knowledge that distinguishes one product, one company, one country from another.

Coming back to my 30-page report, I received the worst possible review. "Huh?!" was the only reply I got from the executive management. I don't blame them. If anyone else was in their shoes, reading a technical report that compares a software to an onion, you could most likely have the same reaction. So why on earth did I make this comparison?

It's simple, because onions do not rot in a way other perishable produce do. The progress of decay in an onion is in layers. If the outside layers get infected by bacteria, the inner layers would be still intact. This is because each layer is carefully isolated and independent from another layer. So if you haven't guessed it by now, I had proposed a new software architecture based on targeted functional layers specific for different applications.

You might ask now how the Layers architecture works? The concept of layers is to reduce dependencies between various engineering disciplines involved in product development while keeping the functionality of the software intact. Layers also help to break the development constraints in a multiplatform product consisting of hardware and software. As we know, hardware and software follow their own life cycle at their own pace. Often we see that either the hardware gets a head start while software waits for the hardware development to complete, or often the software becomes much more complex than its own hardware platform. In either case, software is always late, incomplete, and buggy. Unfortunately this is the case in every embedded systems market that makes the software the bottleneck in our product development processes.

The first golden rule of removing the bottlenecks is to remove dependencies. This is not 20 years ago when we didn't have tools to do hardware in the loop (HIL), software in the loop (SIL), or model in the loop (MIL) simulations. Back then we didn't have evaluation boards handy for every product. We can now simulate the entire product no matter how complex its functionality is. However, you might ask, we have the tools but why aren't our processes still not as efficient? It's because the tools can't think. What we need is a trusted process that would enable us to organize our thoughts systematically and across various engineering disciplines. Layers is a work frame that will enable us to architect our products before they are even formed into a meaningful concepts. It is an organic process solely based on our understanding of our own customer's requirements. It is organic because it is formed based on your application, product, organization, and company's culture. Once we inputted and analyzed the customer's voice, then the developers and the tools can take over and breathe through the development process – from research, to design, to manufacturing, deployment, and beyond.

Nevertheless, this is not the end of the story: there are still various other bottlenecks in the development. Testing is one of those. Product testing is one of the lengthiest and possibly the single most expensive item in the development. In addition, various engineering departments design and perform their own engineering/manufacturing tests independently from one another and often unknowingly overlap in test scopes. Since Layers emphasizes independence, the chances of redundancy would increase even more. However if a common product integration tool is used across all the development, this common language would make the redundancies evident to the designers; therefore they can address it beforehand. But this is not enough because the ultimate goal is to remove lengthy and expensive redundancies in the test in order to make our processes lean.

To create lean processes you can't just jump into buying fancy tools yet, unless you'd like to add to your collection of very expensive dust collectors. We need an active product architecture that is designed based on the customer's specific needs. With the help of the Requirements Model and Model-Based Design tools, creating and maintaining an active product architecture is easier than ever. Furthermore, these tools will enable the development team, from research to manufacturing, to cut back on the amount of documentations without compromising the integrity of the design. For the people who are worried about government regulations and scrutiny of its various branches such as DOD, FDA, DOT, DOE, EPA, ARB, etc., this would provide a documentation system with a robust traceability feature for your design.

Although the “onion” architecture belonged to that particular company facing a specific security problem, the idea of layers can well be expanded into any existing development whether in software, hardware, or a mixture of both, especially in embedded systems. Each department only needs a portion of this software while still receiving the majority of product knowledge. The idea of layers will provide a solution to unify the different languages that now exist in each development segment. It reduces the rework and the time to market, and most importantly saves money, which in return enables the manufacturers to stay competitive not only locally but globally. The macroeconomic impact of deploying intelligent development and product architecture such as Layers would not be dismissive. It is now time to turn away from looking at software as a commodity and see it as a conduit in which knowledge flows.

Finally, by removing the constraints in our development processes, we would be able to implement one Agile framework for development for both hardware and software rather than having a hodgepodge of traditional V-Model or Water Fall in software and ancient phase-gate (CPD)/CPM systems for our hardware. Nevertheless, you still can represent your progress in a phase-gate approach if you choose to, but the development team won’t be bugged down by it as they will follow the Agile approach. All these would empower the development team who is doing the bulk of the work to synchronize their paces across the board while keeping a desirable cadence to deliver incremental values to the customers.

CHAPTER 1

The History of Layers Architecture

What does developing a real-time system project have in common with a construction project, from the project management perspective? Can we develop and manage both projects with the same methodologies and tools? Some of you might think from a project management standpoint, developing real-time or embedded systems should not resemble developing a construction project. Although both might look similar organizationally and share the same types of resources such as time, money, and people; however, the fact that software has become an integral part of embedded systems will significantly differentiate these two projects from each other. Now, this important question arises: why is it that in many companies, both developments are called projects and use the same project management tools? The answer to this question will go beyond the boundaries of what the science of management can offer. This is because the sciences involved with developing real-time systems are very new and are in a constant state of change. Therefore the word "project" might carry a misleading connotation when it's used for developing embedded systems. But this misconception has deeper roots than one might think. We all have seen many leaders in the embedded systems industry still utilize the same tools and methods that a construction company uses in developing their projects.

Nevertheless there are very valuable lessons in studying the construction projects, not in the management methods that have been used for years but in their own schools of thought. The construction industry has evolved through their thousands of years of history. All in all, the traditional management tools can work for real-time systems projects but in no way can one call this type of project management efficient for developing mid- to large-size embedded systems. Through reading this book you will realize how developing embedded systems are fundamentally different from any other types of projects. To start off, let's avoid the use of the term "project" temporarily as it

M. Mirtalebi, *Embedded Systems Architecture for Agile Development*,
https://doi.org/10.1007/978-1-4842-3051-0_1

might misleadingly imply only a process. An embedded system development is not just a process, it is a product integrated into a process. Although you might say a physical building is considered a product but it is not, because it lacks manufacturing of the same building in a volume of thousands of identical copies. Since a building is not a product, therefore, the term "project" shouldn't apply to embedded systems. If you want, you can, but keep in mind, our embedded system development project implies the process and product.

The fact that there will be thousands of copies of what you are making in the market makes the concept of efficiency of grave importance. An efficient development method results in an efficient product, and this translates directly to waste elimination, which creates direct economic impacts on both our company and customers through our process and products respectively. If we reduce the cost of development, our product would become cost effective and more competitive in pricing. In return it will bring a considerable amount of saving of costs to our customers. Combine efficiency and the astronomical numbers of embedded systems used currently across the globe, and we will come to the realization that the efficiency we intend to create will propagate to the furthest corner of the earth. Now this is a true green process. So forget not about the green products, but instead imagine your product will be the greenest of all. However, for some reasons, the idea of green in the industry has been slowly becoming a concept confined only to some promotional tools for marketing biodegradable materials in products or environmentally compliant methods such as ISO14000.

Although the recent waves of the green movement in the waste management and manufacturing process improvements are very good starting points for creating efficient systems, the bulk of waste in resources happens during development of products of any sorts. It is ironic that the term "green," which would imply efficient should result in lower costs; but rather it imposes additional compliance regulations by which the manufacturers must assign budgets to maintain. This will result in higher developmental costs and longer product time to market, which consequently results in higher product prices defeating the purpose of being green. As we can see, a true green process results in better economic impacts such as a lower finished cost of the product without compromising the quality of the product. Therefore our concept of green runs a bit deeper, and it starts from where the first idea of a product sparks; however, it doesn't stop there and it continues to bear fruit while it runs its entire course of product life cycle. The Agile method of development is one way to reduce the waste since these methodologies were initially created to specifically target waste; however there are two shortcomings in

these frameworks. One is that most Agile methods are optimized for software products, not embedded systems that are comprised of both software and hardware. Also, embedded systems are produced in order to carry out critical applications where general purpose computers are not to be trusted to perform the tasks; therefore saving costs is not the main objective of creating these systems.

In other words, most real-time systems are mainly designed to carry out control functions with critical applications. The Agile methods give waste elimination the highest priority, which could make us happy about the efficiency portion, but they fail to put enough emphasis on the robustness of the product. A good example of this is the Microsoft products that are developed through Agile methods. Their numerous after-release software patches and their daily software updates are evident that robustness is not high on their list. We are not to blame one product over its lack of quality. What we want to say is that, when there is not enough emphasis on one aspect of product or there is so much of it on another aspect, people who follow those methods blindly might not be able to put things in perspective. In addition, since in the span of development time and also product life cycle the technology and hardware components can change, it is very important to plan carefully ahead. While an Agile method empowers the project to deal with short-term changes, it might make it easier for its followers to abandon long-term plans for the product. The real consequence of this approach is that the importance of architecture in product development will be devalued. As you can see later, the product architecture plays a vital role in developing the embedded systems.

A project manager, in any type of project, from construction to hi-tech needs to put several hats on during the life of the project. In a real-time system development project, a manager needs to interface with different stakeholders and internal and external customers at different phases of a project and needs to follow various life cycles including project, product, and software life cycles. However, a project manager is hardly a product architect. This is because most project managers are trained and skillful in the art of management rather than designing a real-time system. This book is intended to find the common aspects of various life cycles and introduce an inclusive methodology that would cover them all under one umbrella with an emphasis on the necessary amount of documentation. We are hoping that by the end by reading this book, if not anything, at least your outlook will change toward real-time systems development.

Furthermore, this book is for the embedded system project architects who are aware of steps involving developing a real-time system and not just the project. The detail of how to become aware of the design steps is through systematic use of a tool called the

requirement model. This new addition to what Agile methods seem to be lacking will empower the project architects to create a robust architecture that would very well address the criticality of a real-time system development while creating an efficient process and product. The advantage of following an Agile method will break the never-ending cycle of analysis-paralysis that most classical product development methods suffer from. Nevertheless, we all know that the classical methods are the foundation of the new methods; therefore, the ideas discussed here might be new but backward compatible and are absolutely true to the traditional methods, especially the CPD framework. Nevertheless, this book will not discuss the details of any design standards; software and hardware testing methods such as white box, black box, or any hardware development phase such as hardware alpha and beta prototyping; and manufacturing bottlenecks such as functional and end-of-the line testing, but it will show you how to utilize these methods and tools when you get to the different phases of development.

Nevertheless, if we look at the nature, there is a great lesson in it, that anything and everything in nature happens for a reason. There is no beauty for the sake of only beauty. Any vibrant or dull color and exotic or subtle shape carries a reason. It is useless to talk about team structure and function if we don't know the mission. A mission creates a structure and a structure delivers a function. As Louise Henry Sullivan, the father of skyscrapers and the founder of the school of organic architecture said, "Form Follows Function." This book is to show you how to build your processes around the product.

The New and the Old

If you are coming from an Industrial Engineering background, you know that a simple Gantt chart would never satisfy you if you are planning a project. This is because you believe a project must have a baseline. But why don't most project managers create project baselines? It is because it's a tedious job as you have to break down the project to the units of work per person, which are called tasks, then you have to go on to load all the tasks with resources that they are required to have their own work calendars. Finally you have to apply the logical and temporal constraints to see how the tasks line up, and at the end you have to level the resources by sliding the tasks in the schedule. Therefore, creating a simple Gantt chart by no means is project planning and control. Years later when I changed gears in my career and became an electrical engineer, to my astonishment I saw the embedded system development projects were managed the same way as a construction project was managed.

However, considering the short history of electronics, software, and computer engineering relative to other branches of engineering such as mechanical, chemical, and civil, these new disciplines are still at their infancy stage with respect to work standardization and offering best practices. Nevertheless, our modern manufacturing and the backbone of our industries are built on the foundation of these new branches of engineering. So the storyline goes like this: there are three young sheriffs in our town and they are clueless.

Clash of Cultures

In every traditional development project there are two types of people. People who know the product and people who know the project and hardly enough people who know the product and project at the same time. People who know the product can tell you very well what the product is comprised of. They can identify the sub-assemblies, modules, and components; and they can break down the product into various functions. People who know the project can arrange development and manufacturing work in such way that product or concepts would come to life through the path of the least resistance. However, both groups of people have one thing in common: they are concerned about the constraints. Product-oriented crowds want to establish constraints and project-oriented individuals want to avoid them. This brings up a very interesting phenomenon in product development and manufacturing: the clash of cultures.

Clash of Thoughts

By definition, a project is a unique set of activities that a distinctive team of people carry out only once. On the other hand, a process is what a fixed number of people do routinely. With respect to constraints, projects want to remove the constraints and processes want to solidify them. In classical project planning, the project manager starts drafting the project with no constraints in mind, with the sort of 9-women-giving-birth-to-a-child-in-a-month mentality. Therefore, ideally all project activities start at the same time and continue concurrently. Consequently, the project managers don't want any resource, material, and budget constraints. However, in an engineering process everything is diagonally opposing this view. Engineers, for example, want design reviews before approvals, prototypes before releases, and so forth. This brings us to another fundamental difference between processes and projects; let's call it the clash of school of thoughts.

Projects and Processes

By calling a product development, which inherently has a procedural nature, a project, you bound yourself, your team, and your company to utilize project planning and control methods and tools instead of utilizing methods and tools that were specific to the processes. This is contradictory to the nature of product development, because a product development is much closer in nature to manufacturing, which is also process based, than being a project in its classical form. Let's remember that the most powerful modern product development methodologies such as Lean and Agile come from manufacturing environments but in contrast, the project management methods and tools are originated from the non-manufacturing environments.

Let's assume your company manufactures switching power supplies in various sizes for different applications. Your company has two main lines of products, a fixed line with the highest volume of production; and a custom line that is low in volume but for special applications. You, as an owner, decide to call these two product lines differently. You call the more established product line a process and the custom line or any new product development, a project. In the best case, you decide to use a common pool of resources to support both lines; otherwise you have no option but to have two parallel teams with the same skill sets to work on two different lines. As long as you are using the same resources in the common pool to support and also develop products, you have created a very inefficient system with two very different product visions, project vs. process. But the worst case is when you use two different teams for these two different lines, and then you lose valuable product knowledge in transition from one team to another. In both cases your system has become highly inefficient.

Products and People

A product is nothing more than a collection of constraints that have been materialized. The size, value, scope, and functionality of a product are predefined from day one of its inception. The moment you envision a product in your mind you have constrained it. Consequently the product developers are people who are aware of these constraints and work around or with them to conceptualize the product. Let's say you start a new "project" and invite numerous subject-matter experts into the development team. Let's assume you invited Matt, an expert in power electronics hardware design, to the team.

He's been doing this for the last 15 years. To him, project x, y, and z don't mean much. All he cares about is to design a robust hardware that meets all the constraints or in better words all the requirements of any particular "project."

Product Software

Now the big question is where the software development falls into product development as a whole? Software is becoming a very integral part of our lives and it is growing fast in complexity, which is accepting a bigger portion of the product. The traditional software development methods and life cycles such as V-Model, waterfall, and so forth are project-based methods, because the hallmark of a project is that it has a distinctive start, an end, and some transitional stages in between. The reason V-Model and other software life cycles are so popular now in software development is because when software needed development methods, the only trusted methods available all had roots in project-based methodologies. Software science took off so fast that it left the method thinkers in the dust, giving no advance warning to them to come up with standards and established methods. But now we are at a different juncture of time where we have process-based methodologies such as Lean and Agile, which are gaining tremendous grounds in the industry. This is good news for the software industry but not good enough for the embedded systems.

Embedded Systems

These systems are neither purely hardware nor software but a combination of both. As a matter of fact, the software takes a different name in these systems, called firmware, emphasizing how firmly software is tied to the hardware. The fundamental problem that lays down in the nature of embedded systems is originated from how differently hardware and software are viewed from the developmental standpoint. By adopting two different methods in developing hardware and software separately, we create the clashes of thoughts and cultures resulting in enormous challenges in embedded systems development as how to synchronize the paces of development in software and hardware to minimize the development losses.

In an attempt to overcome this challenge, some cutting-edge industry leaders have decided to adopt two different approaches for hardware and software separately. Process-based methodologies are chosen for the firmware and project-based methods

are chosen for their hardware development. But this, in no way, will help to resolve the initial issue. You still have two different mythologies under one roof for one product while creating additional problems with respect to the clashes of cultures and thoughts. Let's find out what's the root cause of this synchronization problem? To answer this we should look at what the bottlenecks are in the development of the embedded systems.

Process Bottlenecks

In my previous writings, I pointed out two major bottlenecks, firmware latencies, and test applications. For the embedded systems development, these two are the two sides of the same coin. The test bottleneck would be the real issue and the firmware bottleneck would be a consequential one. In other words, firmware development for an embedded system wouldn't cause any bottleneck if the hardware platform was readily available. But to develop the hardware we need firmware. To resolve this situation some people resort to developing their hardware beforehand to give the firmware a head start prior to the "project's" formal kickoff. But that will make your entire product development gravely inefficient, which not only increases the final cost of the product but also lowers the product quality.

The solution might be to utilize any of the in-the-loop methods such as Model in the Loop, Software in the Loop, and Hardware in the Loop, intelligently. We know that the in-the-loop methods have been utilized in the industry for a long time, but why there is still so much waste in our systems? This is because we don't know when exactly to deploy these tools. We have no plan and no road maps as how to resolve the bottleneck issues.

Intelligent Product Development

As we discussed before we should avoid managing our Hi-Tech developmental process like a construction project. But there is one thing we can learn from construction projects: no one in a construction project starts the project until every participating team is aware of the building or site architecture. You might think there are still a lot of embedded systems manufacturers who just do that. That's correct, but soon after the product architecture is in place, the product hardware and software take their own development paths only to synch by enforcing "project" management tools. An active product architecture would give us a road map on when to deploy our synchronization tools.