# Embedded Software Design

## A Practical Approach to Architecture, Processes, and Coding Techniques

Apress®

Jacob Beningo
Linden, MI, USA

ISBN 978-1-4842-8278-6 e-ISBN 978-1-4842-8279-3
https://doi.org/10.1007/978-1-4842-8279-3



This Apress imprint is published by the registered company APress Media, LLC, part of Springer Nature.

The registered company address is: 1 New York Plaza, New York, NY 10004, U.S.A.

*To my family, friends, and mentors.*

## Preface

## Successful Delivery

Designing, building, and delivering an embedded product to market can be challenging. Products today have become sophisticated, complex software systems that sit upon an ever-evolving hardware landscape. At every turn, there is a challenge, whether it's optimizing a system for energy consumption, architecting for configurability and scalability, or simply getting a supply of the microcontrollers your system is built. Embedded software design requires chutzpah.

Despite the many challenges designers, developers, and teams often encounter, it is possible to successfully deliver embedded products to market. In fact, I'll go a step further and suggest that it's even possible to do so on time, on budget, and at the quality level required by the system stakeholders. It's a bold statement, and, alas, many teams fall short. Even the best teams stumble, but it is possible to reach a level of consistency and success that many teams can't reach today.

*Embedded Software Design*, this book, is about providing you and your team with the design philosophies, modern processes, and coding skills you need to deliver your embedded products successfully. Throughout the book, we cover the foundational concepts that will help you and your team overcome modern challenges and help you thrive. As you can imagine, the book is not all-inclusive in that every team will have different needs based on their industry, experience, and goals. *Embedded Software Design* will help get you started, and if you get stuck or need professional assistance, I'm just an email away.

Before we dive into the intricate details of *Embedded Software Design*, we need to initialize our minds, just like our embedded systems. We need to understand the lens through which we will view the material in this book and how we can apply it successfully to our unique situations. This chapter will explore how this book is organized, essential concepts on how to use the material, and how to maximize the value you receive from it.

> **Tip** At the end of each chapter, you will find exercises to help you think through and apply the material from each chapter. "Action Items" will

come in the form of questions, design experiments, and more. To maximize your value, schedule time in your calendar to carefully think through and execute the Action Items.

## The Embedded Software Triad

Many teams developing embedded software struggle to deliver on time, on budget, and at a quality level that meets customer expectations. Successful development is often elusive, and software teams have all kinds of excuses for their failure.[1] However, many teams in the industry are repeatedly successful even under the direst conditions. I've found that these teams have mastered what I call the embedded software triad.

The embedded software triad consists of

- Software Architecture and Design
- Agile, DevOps, and Processes
- Development and Coding Skills

Software architecture and design consists of everything necessary to solicit the requirements, create user stories, and design the architecture. The software architecture defines the blueprint for the system. It's important to note that the architecture should be evolvable and does not necessarily need to be developed up front.[2] How the architecture is designed follows the processes used by the team.

Agile, DevOps, and Processes are the procedures and best practices that are used to step-by-step design, build, test, and deliver the software. Processes provide the steps that allow a team to deliver successfully based on the budget, time, and quality parameters provided to them consistently. In today's software development environment, processes are often based on various agile methodologies. Therefore, the processes help to guide the implementation under the parameters and disciplines defined by the team. I've also called out DevOps specifically because it is a type of process often overlooked by embedded software teams that offers many benefits.

Development and coding skills are required to construct and build the software. Development entails everything necessary to take the architecture and construct the system using the processes! At the end of the day, the business and our users only care about the implementation which is completed through development and coding skills. The implementation should result in a working system that meets agreed-upon features, quality, and performance characteristics. Teams will organize their software structure, develop modules, write code, and test it. Testing is a critical piece of implementation that can't be overlooked that we will be discussing throughout the book.

To be successful, embedded software designers, developers, and teams must not just master the embedded software triad but also balance them. Too much focus on one area will disrupt the development cycle and lead to late deliveries, going over budget, and even buggy, low-quality software. None of us want that outcome. This brings us to the question, "How can we balance the triad?"

## Balancing the Triad

Designers, developers, and teams can't ignore any element of the embedded software triad. Each piece plays a role in guiding the team to success. Unfortunately, balancing these three elements in the real world is not always easy. In fact, in nearly every software audit I have performed from small to large companies, there is almost always some imbalance that can be found. In some teams, it's minor and adjustments can result in small efficiency improvements. In others, the imbalance is major and often crippling to the company.

The relationships between the elements in the triad can be best visualized through a Venn diagram, as shown in Figure 1. When a team masters and balances each element, marked by the number 4 in Figure 1, it is more likely that the team will deliver their software on time, on budget, and at the required quality level. Unfortunately, finding a team that is balanced is relatively rare. It is more common to find teams that are hyperfocused on one or two elements. When a team is out of balance, there are three regions of Figure 1 where the teams often fall, denoted by 1, 2, and 3. Each area has its own symptoms and prescription to solve the imbalance.


***Figure 1*** The balancing act and results caused by the embedded software triad

> **Caution**Being out of balance doesn't ensure failure; however, it can result in the need for herculean efforts to bring a project together successfully. Herculean efforts almost always have adverse effects on teams, costs, and software quality.

The first imbalance that a team may fall into is region 1. Teams fall into region 1 when they focus more on software architecture and implementation (denoted by a 1 in Figure 1) than on development processes. In my experience, many small- to medium-sized teams fall within region 1. In smaller teams, the focus is almost always delivery. Delivery equates to implementation and getting the job done, which is good, except that many teams jump right in without giving much thought to what they are building (the software architecture) or how they will implement their software successfully (the processes) repeatedly and consistently.[3]

Teams that operate in region 1 can be successful, but that success will often require extra effort. Deliveries and product quality will be inconsistent and not easily reproducible. The reason, of course, is that they lack the processes that create consistency and reproducibility. Without those processes, they will also likely struggle with quality issues that could cause project delays and cause them to go over budget. Teams don't necessarily need to go all out on their processes to get back into balance, but only need to put in place the right amount of processes to ensure repeatability.

> **Tip**If you spend 20% or more debugging your software, you will most likely have a process problem. Don't muscle through it; fix your pro-

cesses!

The second unbalanced region to consider is region 2. Teams that fall into region 2 focus on development processes and implementation while neglecting the software architecture (denoted by a 2 in Figure 1). These teams design their system on the fly without any road map or blueprint for what it is they are building. As a result, while the team's software quality and consistency may be good, they will often still deliver late because they constantly must rework their system with every new feature and requirement. I often refer to these teams as lost or meandering because they don't have the big picture to work from.

The final unbalanced region to consider is region 3. Teams in region 3 focus on their software architecture and processes with little thought given to implementation (denoted by a 3 in Figure 1). These teams will never complete their software. They either lack the implementation skills or bog down so much in the theory of software that they run out of money or customers before the project is completed.

There are several characteristics that a balanced team will exhibit to master the embedded software triad. First, a balanced team will have a software architecture that guides their implementation efforts. The architecture is used as an evolving road map that gets the team from where they are to where their software needs to be. Next, a balanced team will have the correct amount of processes and best practices to ensure quality software and consistency. These teams won't have too much process and not too little. Finally, a balanced team will have development and coding skills to construct the architecture and leverage their processes to test and verify the implementation.

## Successful Embedded Software Design

Successful software delivery requires a team to balance the embedded software triad, but it also requires that teams adopt and deploy industry best practices. As we discuss how to design and build embedded software throughout this book, our focus will discuss general best practices. Best practices "are procedures shown by research and experience to produce

optimal results and establish or propose a standard for widespread adoption."[4] The best practices we focus on will be general to embedded software, particularly for microcontroller-based systems. Developers and teams must carefully evaluate them to determine whether they fit well within their industry and company culture.

Applying best practices within a business requires discipline and an agreement that the best practices will be adhered to no matter what happens. Too often, a team starts following best practices, but when management puts on the heat, best practices go out the window, and the software development process decays into a free for all. There are three core areas where discipline must be maintained to successfully deploy best practices throughout the development cycle, as shown in Figure 2.


***Figure 2*** Best practice adoption requires discipline, agreement, and buy-in throughout a business hierarchy to be successful

Developers form the foundation for maintaining best practices. They are the ones on the frontline writing code. If their discipline breaks, no matter what else is going on in the company, the code will descend into chaos. Developers are often the ones that also identify best practices and bring new ideas into the company culture. Therefore, they must adhere to agreed-upon best practices no matter the pressures placed upon them.

Developers may form the foundation for adhering to best practices, but the team they work on is the second layer that needs to maintain discipline. First, the team must identify the best practices they believe must be followed to succeed. Next, they need to reinforce each developer, and the team needs to act as a buffer with upper management when the pressure

is on. Often, a single developer pushing back will fail, but an entire team will usually hold some sway. Remember, slow and steady wins the race, as counterintuitive as that is to us intellectually and emotionally.

> **Definition**Best practices "are procedures shown by research and experience to produce optimal results and establish or propose a standard for widespread adoption."[5]

Finally, the management team needs to understand the best practices that the team has adopted. Management must also buy into the benefits and then agree to the best practices' value. If they know why those best practices are in place, they will realize that it is to preserve product quality, minimize time to market, or some other desired benefit when the team pushes back. The push to cut corners or meet arbitrary dates will occur less but won't completely go away, given its usually market pressure that drives unrealistic deadlines. However, if there is an understanding about what is in the best interest of the company and the customers, a short delay for a superior product can often be negotiated.

## How to Use This Book

As you might have guessed from our discussions, this book focuses on embedded software design best practices in each area of the embedded software triad. There are more best practices within the industry than could probably be written into a book; however, I've tried to focus on the best practices in each area that is "low-hanging fruit" and should provide high value to the reader. You'll need to consult other texts and standards for specific best practices in your industry, such as functional safety.

This book can either be read cover to cover or reviewed on a chapter-by-chapter basis, depending on the readers' needs. The book is broken up into four parts:

- Software Architecture and Design
- Agile, DevOps, and Processes
- Development and Coding Skills
- Next Steps and Appendixes

I've done this to allow the reader to dive into each area of the embedded software triad and provide them with the best practices and tools necessary to balance their development cycles. Balancing can be sometimes accomplished through internal efforts, but often a team is too close to the problem and "drinking the Kool-Aid" which makes resolution difficult. When this happens, feel free to reach out to me through www.beningo.com to get additional resources and ideas. *Embedded Software Design* is meant to help you be more successful.

A quick note on the Agile, DevOps, and Processes part. These chapters can be read independently, but there are aspects to each chapter that build on each other. Each chapter walks you through getting a fundamental piece of a CI/CD pipeline up and running. For this reason, I would recommend reading this part in order.

I hope that whether you are new to embedded software development or a seasoned professional, you'll find new or be reminded of best practices that can help improve how you design and develop embedded software. We work in an exciting industry that powers our world and society can't live without. The technologies we work with evolve rapidly and are constantly changing. Our opportunities are limitless if we can master how to design and build embedded software effectively.

**Action Items**

To put this chapter's concepts into action, here are a few activities the reader can perform to start balancing their embedded software activities:

- Which area do you feel you struggle the most in?
  - Software Architecture and Design
  - Agile, DevOps, and Processes
  - Development and Coding Skills
- Review Figure 1. Which region best describes you? What about your team? (If you are not in region 4, what steps can you take to get there?)
- Take Jacob's design survey to understand better where you currently are and how you can start to improve (www.surveymonkey.com/r/7GP8ZJ8).

- Review Figure 2. Do all three groups currently agree to support best practices? If not, what can be done to get the conversation started and get everyone on the same page?

## Acknowledgments

The book you now hold in your hand, either electronically or in paper form, is the product of 12 months of intense writing and development. The knowledge, techniques, and ideas that I share would not have been possible without over 20 years of support from family, friends, mentors, and teachers who helped me get to this point in my career. I humbly thank them for how they have enriched my life and pushed me to be the best version of me.

I would also like to thank all of you who have corresponded with me and even been my clients over the years. Your thoughts, suggestions, issues, challenges, and requests for help have helped me hone my skills and grow into a more mature embedded software consultant. I've learned how to more effectively help others, which has helped me continue to provide more value to my clients and the embedded systems industry.

I owe a big thank you to Jack Ganssle for reviewing this book. I know a lot of the review occurred during the summer while he was sailing the globe. That didn't stop him from asking great questions and giving feedback that provided me with new insights, ideas, and guidance that helped me dramatically improve the book's content.

There are also many, many others who have helped to bring this book into existence. The acknowledgments could go on indefinitely, but instead I'll simply say thank you to Apress, Steve Anglin, James Grenning, Jean Labrosse, John Little, James McClearen, Mark Powers, and Tomas Svitek.

# Table of Contents

## About the Author

**Jacob Beningo**

is an embedded software consultant with around 20 years of experience in microcontroller-based, real-time embedded systems. Jacob founded Beningo Embedded Group in 2009 to help companies modernize how they design and build embedded software through software architecture and design, adopting Agile and DevOps processes and development skills. Jacob has worked with clients in over a dozen countries to dramatically transform their businesses by improving product quality, cost, and time to market in the automotive, defense, medical, and space systems industries. Jacob holds bachelor's degrees in Electrical Engineering, Physics, and Mathematics from Central Michigan University and a master's degree in Space Systems Engineering from the University of Michigan.

Jacob has demonstrated his leadership in the embedded systems industry by consulting and training at companies such as General Motors, Intel, Infineon, and Renesas, along with successfully completing over 100 embedded software consulting and development projects. Jacob is also the cofounder of the yearly Embedded Online Conference, which brings together managers and engineers worldwide to share experiences, learn, and network with industry experts.

Jacob enjoys spending time with his family, reading, writing, and playing hockey and golf in his spare time. In clear skies, he can often be found outside with his telescope, sipping a fine scotch while imaging the sky.

## About the Technical Reviewer

**Jack Ganssle**

has written over 1000 articles and six books about

embedded systems, as well as a book about his sailing fiascos. He started developing embedded systems in the early 1970s using the 8008. He's started and sold three electronics companies, including one of the bigger embedded tool businesses. He's developed or managed over 100 embedded products, from deep-sea navigation gear to the White House security system… and one instrument that analyzed cow poop! He now lectures and consults about the industry and works as an expert witness in embedded litigation cases.

## Footnotes

1 Yes, I know, a bit harsh right off the bat, but it's necessary.

2 There are many arguments about this based on the processes used. We will discuss further later in the book.

3 We often aren't interested in a one-hit wonder delivery. A team needs to consistently deliver software over the long term. Flukes and accidental success don't count! It's just dumb luck.

4 **www.merriam-webster.com/dictionary/best%20practice**

5 **www.merriam-webster.com/dictionary/best%20practice**