Embedded Systems Roadmaps

Nathan Jones • November 9, 2023

INTRODUCTION

What does it mean to be an "embedded systems engineer"? What skills would it be beneficial for you to develop if you want to be a successful one? Whether you've only just finished your first Arduino project or you've been building embedded systems for decades, we could all benefit from periodically assessing our skills against a baseline and identifying new skills that we can learn. What follows are a handful of opinions about the skills that should be considered as part of a core body of embedded systems or software engineering knowledge. I want to emphasize the word "opinions" because there is no *one* answer to what an embedded systems engineer should know (nor is there, really, for any field). Nonetheless, these "opinions" can be useful to identify the things "we don't know that we don't know", helping make us more well-rounded engineers. (I feel like I experience serendipity all the time in my professional career, by pursuing things of their own accord to make myself a better engineer and then, surprise!, it's needed on a project on which I'm working.) I hope you enjoy this list!

ACM CURRICULUM GUIDELINES FOR UNDERGRADUATE DEGREE PROGRAMS IN COMPUTER ENGINEERING (2016)

The Association of Computing Machinery periodically provides recommendations to undergraduate schools about what to include in certain degree paths and a section of their recommendations from 2016 for a computer engineering degree includes skills and knowledge related to embedded systems. Starting on page 80 of the PDF linked above, the ACM provides a list of course objectives applicable to a one-semester course on embedded systems, which includes the following subject areas:

CE-ESY	Embedded Systems
	[40 core hours]
CE-ESY-1	History and overview [1]
CE-ESY-2	Relevant tools, standards, and/or engineering constraints [2]
CE-ESY-3	Characteristics of embedded systems [2]
CE-ESY-4	Basic software techniques for embedded applications [3]
CE-ESY-5	Parallel input and output [3]
CE-ESY-6	Asynchronous and synchronous serial communication [6]
CE-ESY-7	Periodic interrupts, waveform generation, time measurement [3]
CE-ESY-8	Data acquisition, control, sensors, actuators [4]
CE-ESY-9	Implementation strategies for complex embedded systems [7]
CE-ESY-10	Techniques for low-power operation [3]
CE-ESY-11	Mobile and networked embedded systems [3]
CE-ESY-12	Advanced input/output issues [3]
CE-ESY-13	Computing platforms for embedded systems

This is your typical "intro to embedded systems" course, with the goal being to introduce a student to their first microcontroller, which they will program in an IDE with either the vendor HAL or peripheral registers. They are taught what an embedded system is, how they are programmed and built, and how to use the peripherals on the microcontroller such as GPIO, timers, interrupts, UART, SPI, and I2C. Some courses will also discuss the basics of multitasking, RTOSes, networking, or other useful concepts.

In fact, the other courses listed by the ACM as part of an undergraduate degree in computer engineering (shown below) also seem applicable to a course of study for an embedded systems engineer.

CE-CAE	Circuits and Electronics
CE-CAL	Computing Algorithms
CE-CAO	Computer Architecture and Organization
CE-DIG	Digital Design
CE-ESY	Embedded Systems
CE-NWK	Computer Networks
CE-PPP	Preparation for Professional Practice
CE-SEC	Information Security
CE-SGP	Signal Processing
CE-SPE	Systems and Project Engineering
CE-SRM	System Resource Management
CE-SWD	Software Design

Recommended math courses include

- Analysis of Continuous Functions,
- Discrete structures,
- Linear algebra, and
- Probability and statistics.

The PDF linked above lists recommended course objectives for each of the above courses. A few excellent books (based solely on my own opinion) that discuss these introductory concepts are listed below.

- Making Embedded Systems (Elecia White)
  - One of the few (only?) introductory books I know of that's truly MCU-agnostic
  - Note: A second edition is set to be published in March 2024
- Programming Embedded Systems (Michael Barr)
- AVR Programming (Elliot Williams)
- Embedded Systems Architecture (Daniele Lacamera)

EMBEDDED SKILLS TREE (ELECIA WHITE)

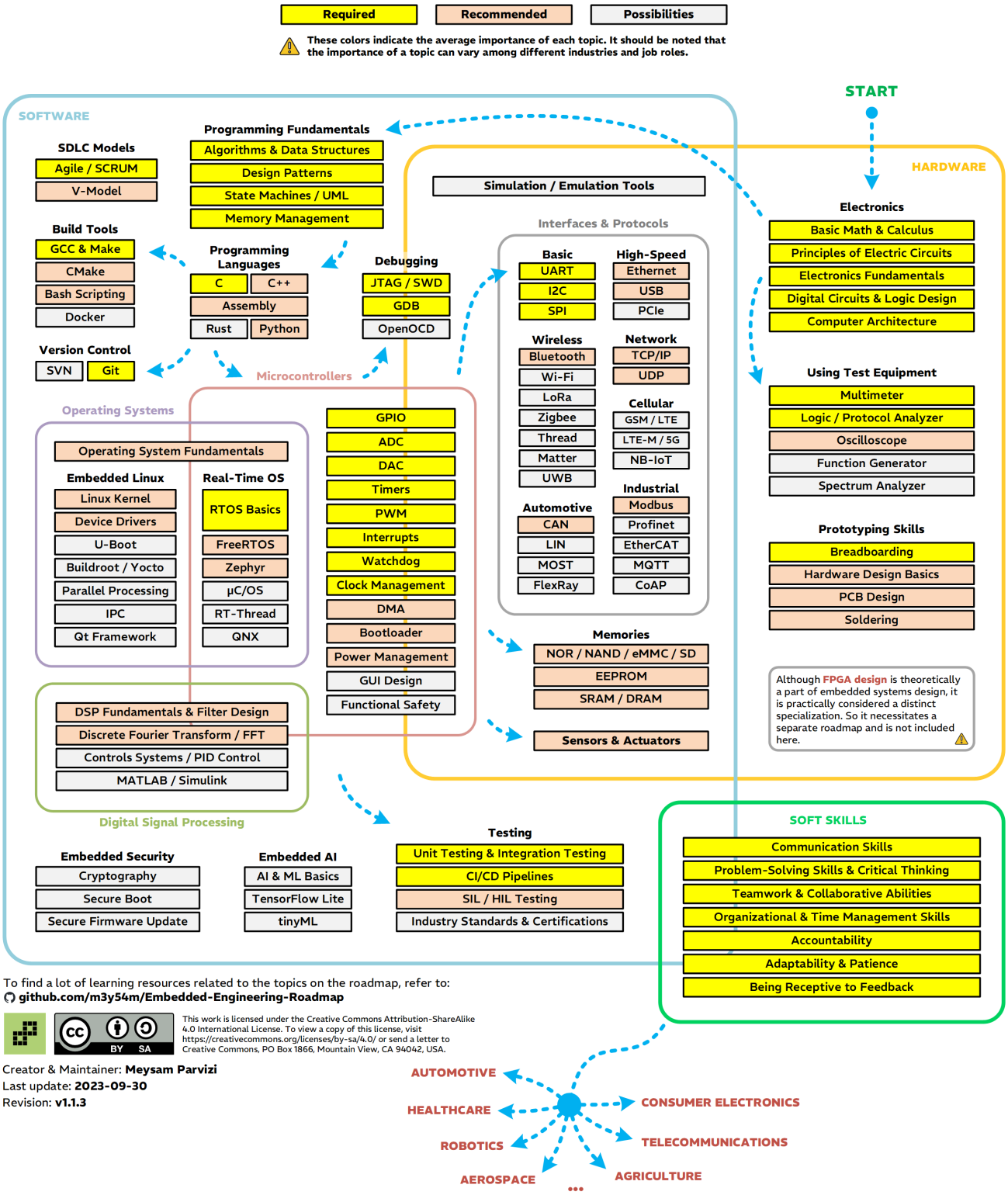
Elecia White, host of the Embedded.fm podcast and author of Making Embedded Systems, has written an embedded systems skills tree using a neat hexagonal chart.

The idea is that you start at the bottom, with "Get an Arduino", and once that skill or task has been completed, you move on to any other task that shares an edge with that task and repeat. The chart expands outwards and upwards and is set up to encourage multiple different paths to achieving mastery.

The list is broad and it includes both skills that might fall into an introduction to embedded systems course ("Configure a timer", "Implement a state machine", "Implement MCU sleep in a low-power project") and also more advanced skills that commercial embedded systems projects might need ("Write a bootloader", "Use a multi-core processor", "Create a CI/CD system").

EMBEDDED SYSTEMS ENGINEERING ROADMAP (MEYSAM PARVIZI)

Meysam Parvizi [m3y54m] is a GitHub user who's created an embedded systems roadmap that covers everything from using a multimeter to understanding CAN.

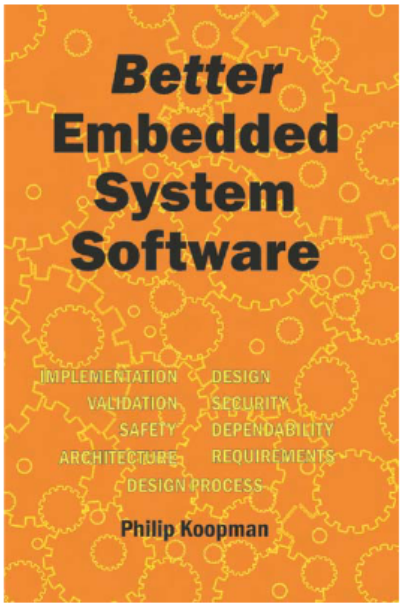


It's a thorough list, though the idea that certain skills are "required" should *maybe* be taken with a grain of salt. When this roadmap was discussed on the Embedded.fm Slack I also remember some folks thinking that the application areas (automotive, aerospace, agriculture, etc) should have been given greater emphasis. Like many fields, having a broad skill set can

make you marketable and useful on many different projects but it's the specific skills that are needed to accomplish the task at hand (which can vary greatly from field to field and even project to project) that can make you invaluable to a specific team.

BETTER EMBEDDED SYSTEMS BY PHILIP KOOPMAN

In 2010, Philip Koopman, a long-time embedded systems engineer who specializes in safety-critical systems, published a book that described all of the ways in which embedded systems designs tend to fail their design reviews, including discussions of everything from software architecture to the use of mutexes to designing for reliability.

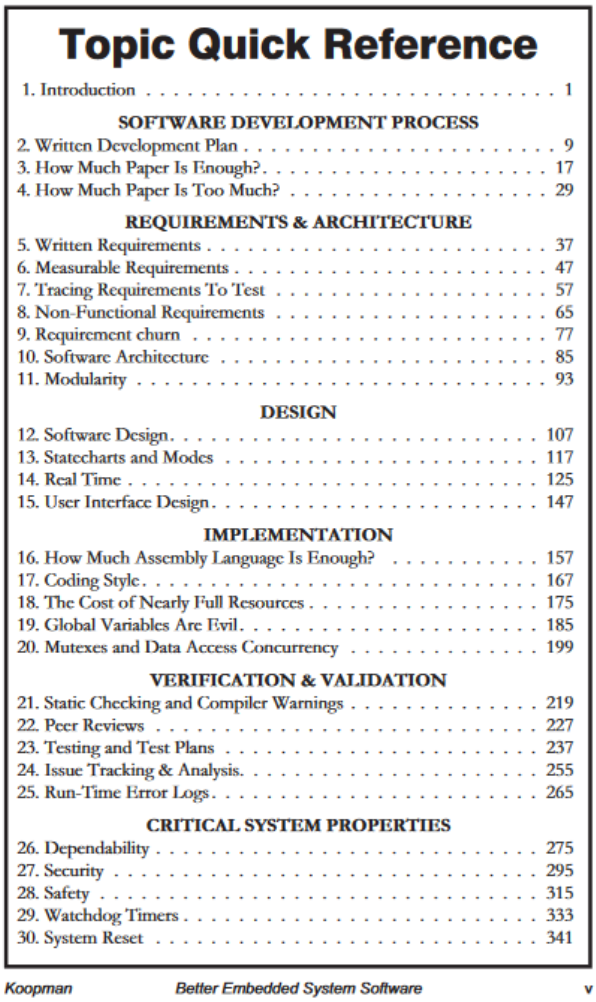


***Better* Embedded System Software**

Philip Koopman
Carnegie Mellon University

Drumnadrochit Education LLC, 2010
Hardcover, 397 pages, acid free paper
ISBN-13: 978-0-9844490-0-2
ISBN-10: 0-9844490-0-0

http://koopman.us/book.html



Topic Quick Reference

I've sometimes described this book as "showing you everything you have yet to learn" about embedded systems, by which I mean it pretty well encompasses the skills (beyond those covered in an introductory course) that someone might need in their career to build and produce high-quality embedded systems. It makes an excellent follow-up to an introductory course, especially if one uses it to generate further study in one or more of the book's subject

areas. The treatment of all but a few topics is, unfortunately and expectedly, a little sparse and my biggest complaint is that Phillip doesn't really provide many references for further research (even though each chapter ends with a "Resources" section). Also, the book doesn't discuss hardware much at all, as could have been predicted by the book's title.

SOFTWARE ENGINEERING BODY OF KNOWLEDGE (IEEE COMPUTING SOCIETY)

Not to be outdone by the ACM, the IEEE Computing Society has also produced a guide to their definition of a "software engineering body of knowledge" for several years. Their guide (available here) lists 15 knowledge areas.

Table I.1. The 15 SWEBOK KAs
Software Requirements
Software Design
Software Construction
Software Testing
Software Maintenance
Software Configuration Management
Software Engineering Management
Software Engineering Process
Software Engineering Models and Methods
Software Quality
Software Engineering Professional Practice
Software Engineering Economics
Computing Foundations
Mathematical Foundations
Engineering Foundations

The knowledge areas are fairly comprehensive (reminiscent of the ACM guidelines above) but, like "Better Embedded System Software", the list doesn't really include any hardware considerations.

SOFTWARE SYSTEMS ENGINEERING PROGRAMMES: A CAPABILITY APPROACH (LANDWEHR, ET AL.)

In 2017, an octet of software engineering researchers (that included famed software engineer and researcher David Parnas) published a paper that discussed the capabilities that they believed software engineers needed to be successful in their jobs.

1. Communicate precisely between developers and stakeholders
2. Communicate precisely among developers
3. Design human-computer interfaces
4. Design and maintain multi-version software
    - Identify and separate changeable concerns
    - Document to ease revision
    - Use parameterization
    - Design software that can be moved to many platforms
    - Design software that is easily extended or contracted
    - Design and maintain products that will be offered in many versions
5. Design software for reuse
6. Ensure that software products meet quality standards
7. Develop secure software
8. Create and use models in system development
9. Specify, predict, analyze and evaluate performance
10. Be disciplined in development and maintenance
11. Use metrics in system development
12. Manage complex projects

The authors state that many current computer science degrees don't teach these skills because computer science is a "research field" while software engineering is a "class of professions". Additionally, they claim that the capabilities listed are fundamental to the discipline of software engineering and, therefore, somewhat timeless. I think it's a good list, particularly in that it seems to acknowledge the skills most often needed for building successful embedded systems "in the wild", more than merely being able to understand or design them.

"EMBEDDED SYSTEMS IS TOO BROAD; THERE'S NO WAY TO CREATE A SINGLE LIST OF CORE SKILLS THAT EVERY EMBEDDED SYSTEMS ENGINEER IS SUPPOSED TO KNOW!" OR "I'VE BEEN AN EMBEDDED SYSTEMS ENGINEER FOR OVER 40 YEARS AND I'VE NEVER NEEDED X" AND RELATED ~~COMPLAINTS~~ COUNTERARGUMENTS

Regarding the former, I think this statement is patently false. Embedded systems doesn't have a monopoly on being a "very broad discipline"; computer science is equally broad and the field of medicine is arguably much, much broader! Both of those fields somehow manage

to identify a core set of required knowledge for every practitioner (as evidenced by their respective degree paths) and I think it's possible for embedded systems to do the same. Even this paper, which seems to advocate for embedded systems education to look like apprenticeships, admits that there are certain skills which are commonly lacking in entry-level embedded systems engineers and which, therefore, would be good to include in an introductory course of study. Additionally, although the specific tools that are used might change, I think it *is* possible to enumerate a set of skills that every embedded systems engineer ought to know. The exact nature of "debugging" might look different from project to project, but I would argue that *all* embedded systems engineers need to be good debuggers, for example.

Further, if embedded systems really doesn't have a core skill set and if it's really best taught like an apprenticeship then we've begun to relegate it more to the realm of "trade" than "engineering discipline". The emergence of coding "boot camps" is the result of a similar push in the realm of computer science, I think, and it seems that although this helps create more people to fill the job of writing software it doesn't produce people who can necessarily write *good* software or, maybe more fairly, *architect good software systems*. With the rise of software and embedded systems in financial systems, avionics, and automobiles, I think there's an imperative to not just make *working* embedded systems but *robust and secure* embedded systems, qualities that don't necessarily arise when something is taught merely as a trade. On a related note, and speaking to the second counterargument above, I think it's to our own benefit to establish and have a broad skill set. Although developing an embedded system based on an 8-bit microcontroller is vastly different from developing one for an application processor running embedded Linux, I would argue that the embedded systems developer who specializes in programming the former and who has *never* programmed the latter is doing themselves a disservice. Although it's definitely *possible* to have a long and successful career as an embedded systems engineer without knowing certain skills that others would consider important, I believe that learning those skills helps make one a more well-rounded engineer and improves their work in the areas where they do most of their work.

For these reasons, I think that embedded systems engineers *can* and *should* define a core skill set that all embedded systems engineers should strive to know.

WRAPPING UP

First, take a breath! You might be feeling overwhelmed right now about how much you don't know and I'm here to tell you that *it's okay to have lots left to learn*! If you enjoy this field then think of it like being a master craftsman: there is joy in getting better, regardless of how close

or far away the goal seems to be. Then, pick an area above that you don't know much about and start learning! Do an internet search for a term you aren't familiar with, pick up one of the books or PDFs I linked, or come up with a fun project you can complete that will require a skill you don't know how to do yet. I wish you all the best on your journey!

Do you have a favorite embedded systems roadmap or topic that wasn't listed? Do you have a favorite book about embedded systems that other folks should know about? Then you can also sound off in the comments to help make our community better!

Previous post by Nathan Jones:
What does it mean to be 'Turing complete'?