

Establishing Great Software Development Process(es) for Your Organization

By Dale Mayes

DMayes@HomePortEngineering.com

**Class: ETP-410
Embedded Systems Conference
San Francisco 2005**

Abstract:

There are many correlations between good software designs and good software development processes. For instance, although there isn't one design that is applicable to every project, there are principles of good designs that are. The same is true in development processes. Also, it takes multiple views to adequately describe both software architectures and software development processes. One of the process views we will examine is a swim lane chart. This view highlights another parallel; good software designs limit the number of dependencies and interaction points between modules. Your process should as well.

We'll start with an introduction to the traditional waterfall, incremental and iterative development models. A quick overview of a classic formal process (DoD-2167a) will establish a common reference for what artifacts are typically created under a formal process. Other factors that add to a project's complexity, and consequently, the need for a more formal process will be reviewed.

Introduction:

On the continuum of software processes where does your company fall? Unless you are developing a one-off throw away project where you are; the customer, the software engineer, and the hardware engineer hopefully you are not working to the extreme left of the spectrum.

No process

DoD-2167a

Even if your company does not have a formal development process for their embedded software, you can probably identify with some of the following typical process steps.

- A customer (or marketing) has a concept for the 'next great' thing.
- The rough customer concepts get refined into robust system level requirements including details on, performance and the desired user interface experience.
- The system requirements are supplemented with any other external obligations (industry standards, safety agencies, company policies...)
- The system architecture is selected addressing:
 - Types of I/O needed
 - Number of boards
 - Type of processor(s)
 - Data and program memory needs
 - User interface
- The system requirements are allocated to the configuration items (hardware boards and microcontrollers)
- The system requirements that were allocated to software are supplemented with additional interface requirements imposed by the hardware or the multiple software processors. In other words, if you have two processors in your design, how will the processors communicate with each other (protocol) and what data will they exchange.
- For each processor, a software team should create the software architecture, identifying the major routines and modules. The following should be estimated:
 - Allocate the processor's total memory budgets (program and data space) across each module.
 - Estimate timing budgets (both real time and cycles consumed) across each module.
 - Validate I/O plans
- The software engineer creates a detailed design of the new software modules and has them reviewed by his peers.

Establishing Great Software Development Process(es) for Your Organization

- The software is then coded following the established 'house' coding standards.
- The software developers perform unit testing of their code.
- Someone other than the software developers verifies the fully integrated software.
- A final validation is performed against the initial system requirements.

This typical development sequence correlates pretty well to the classic waterfall methodology. One development task flows into the next task.

Traditional Waterfall, Incremental and Iterative design methodologies

The waterfall model breaks the development process into a series of cascading phases. The analysis and design phases consist of a top-down decomposition of the system. The implementation, integration and test phases represent a bottom-up construction. The waterfall model is frequently represented in a V shape, see figure 1. Down the left side of the V is the top-down decomposition of requirements and design. Where the system requirements are identified and allocated across the hardware and software components or subsystems. Up the right side of the V is the bottom-up construction of the system, encompassing; build, integrate and test. A nice feature of the V representation is that the testing path flows across the V. This represents two things. First, that tests and test plans can be developed in parallel to the implementation. Second, that there are multiple levels of requirements to be tested as the subsystems get integrated. The requirements directly across the V are what should be verified as the subsystems are integrated and ultimately the full system is built.

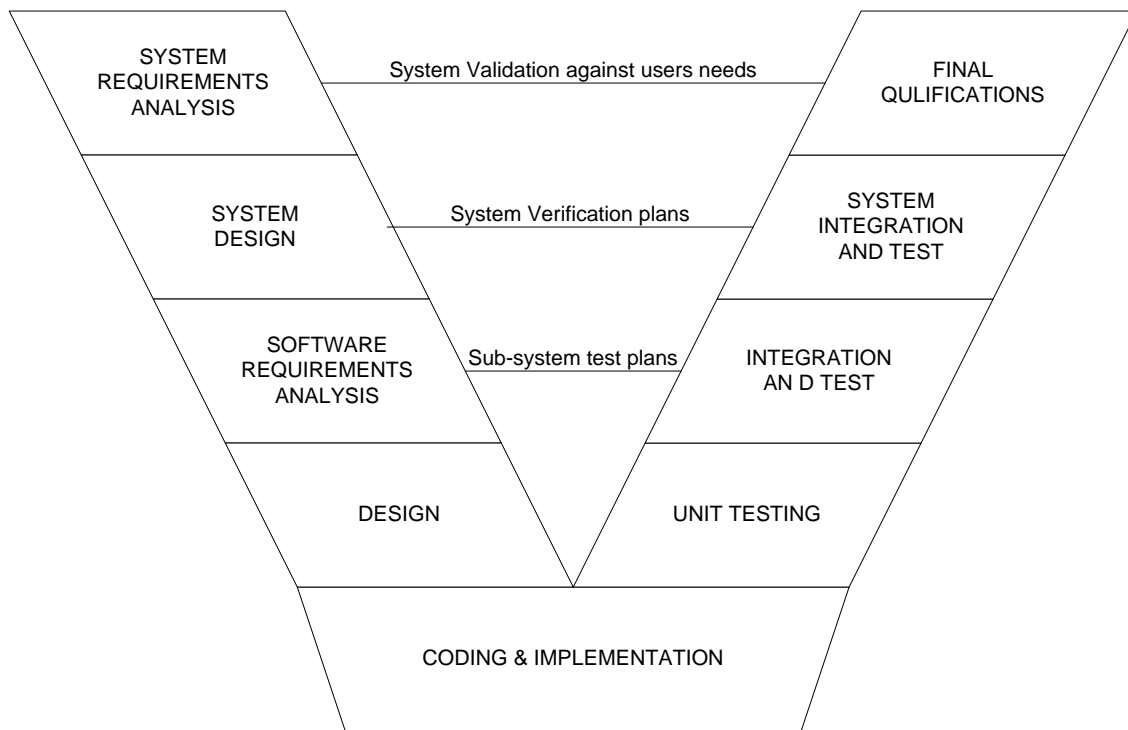


Figure 1 - Waterfall or V model

Establishing Great Software Development Process(es) for Your Organization

As the name implies, the waterfall model assumes a one-way cascading progression of tasks. In other words, it assumes all requirements can be fully defined prior to moving on to design and the system can be fully designed prior to implementations. The waterfall model assumes parallel development of multiple components that make up a system. For example, if the system design phase identifies three microprocessors in the system, there will actually be three simultaneous 'software requirement analysis' phases. These parallel development phases will merge back into one development path after each component completes its 'integration and test' phase.

A major criticism of the waterfall approach is that it doesn't handle downstream changes or incompleteness well. It simplifies the development process into orderly progression of fully attainable tasks. Another criticism of the waterfall method is that it requires significant development up front. Sometimes this method is referred to as BUFD (Big Up Front Design).

One method to overcome this large up front development effort is to incrementally implement the system. In essence, an incremental development runs through the full waterfall cycle for as much of the system as is defined, with the idea that a future pass will add the other functionality. The glossary in "Software Requirements Engineering, second edition", published by IEEE in 1997 has the following definition of Incremental development: "A software development strategy that involves the process of constructing a partial implementation of a total system by slowly adding increased functionality or performance. The initial system is developed through all the system lifecycle stages, from analysis to implementation. This approach reduces the cost incurred before an initial capability is achieved. It also produces an operational system more quickly and thus reduces the possibility that the user's needs will change during the development process."

The iterative development models represent another metaphor for the incremental development approach, see figure 2. The iterative model is typically represented by a spiral expanding out from the origin of the x and y-axis to a 100% endpoint.

The theory behind the iterative model is that 100% of the requirements cannot be known up front and it is only after an implementation that additional requirements are revealed. The four axis's in figure 2 include; requirements, design, code, test. The example in figure 2 represents three iterations to realize a completed system. The first iteration includes the front end system design and implementation of the low level software. This first iteration of software enables the hardware team to test the initial prototypes. The second iteration gets the basic functionality of the product running. The final iteration ensures all the cross-mode interaction, special features, and advanced user interface is fully functioning.

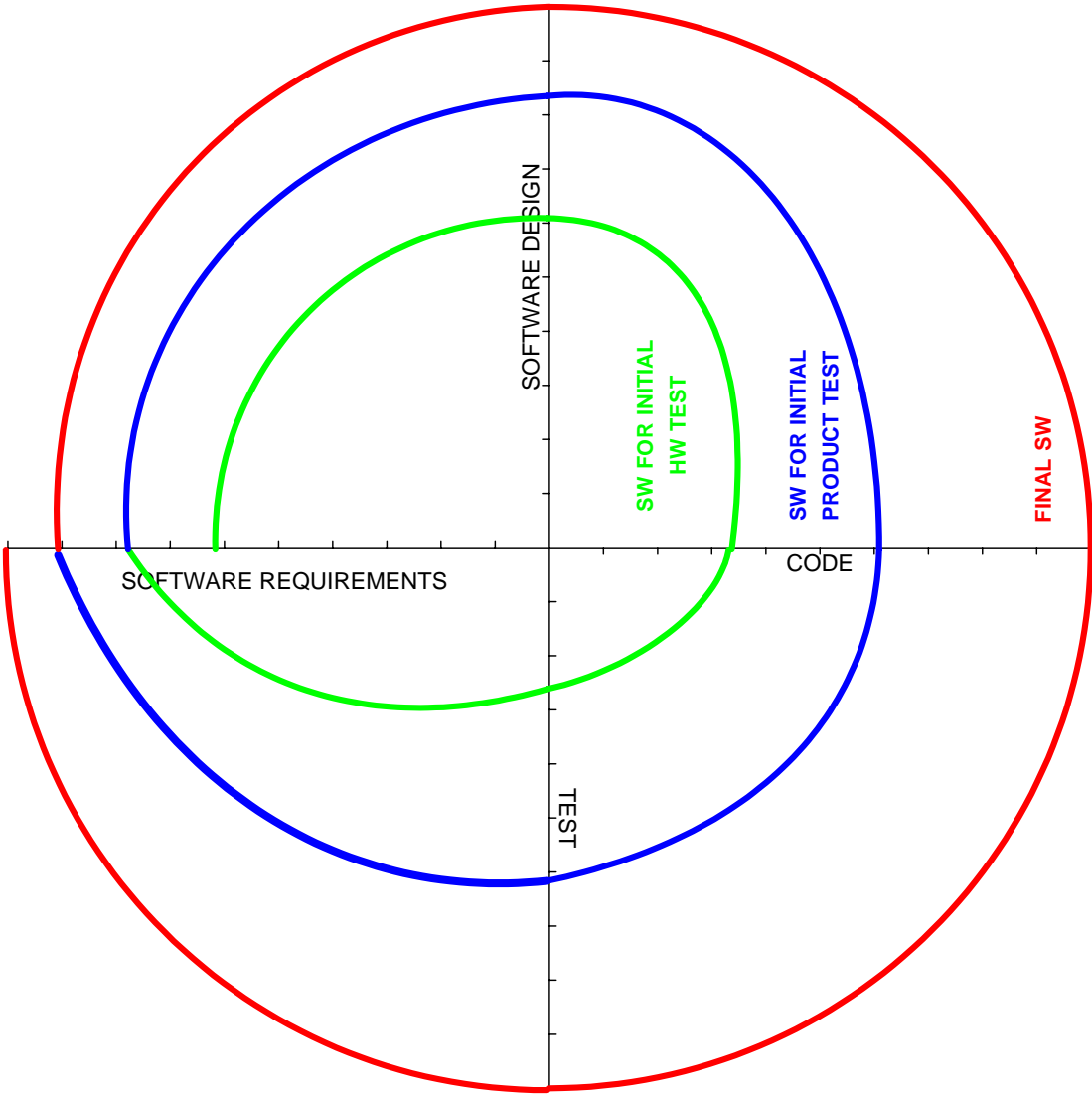


Figure 2 - Iterative Software Development Model

DoD-2167a

DoD-2167a, "Military Standard, Defense System Software Development" was released 29 February 1988. As stated in the forward of the document, "This standard establishes uniform requirements for software development that are applicable throughout the system life cycle. The requirements of this standard provide the basis for Government insight into a contractor's software development, testing, and evaluation efforts."

As shown in figure 3, DoD-2167a decomposes a System into Segments, where a Segment consists of one or more Hardware Configuration Items, one or more Computer Software Configuration Items and an Interface Requirements Specification. Typically, a Hardware Configuration Item is a printed circuit board and a Computer Software Configuration Item is a microcontroller. A Computer Software Configuration Item is further decomposed into Computer Software Components (source code files) and then finally Computer Software Units (functions or modules).

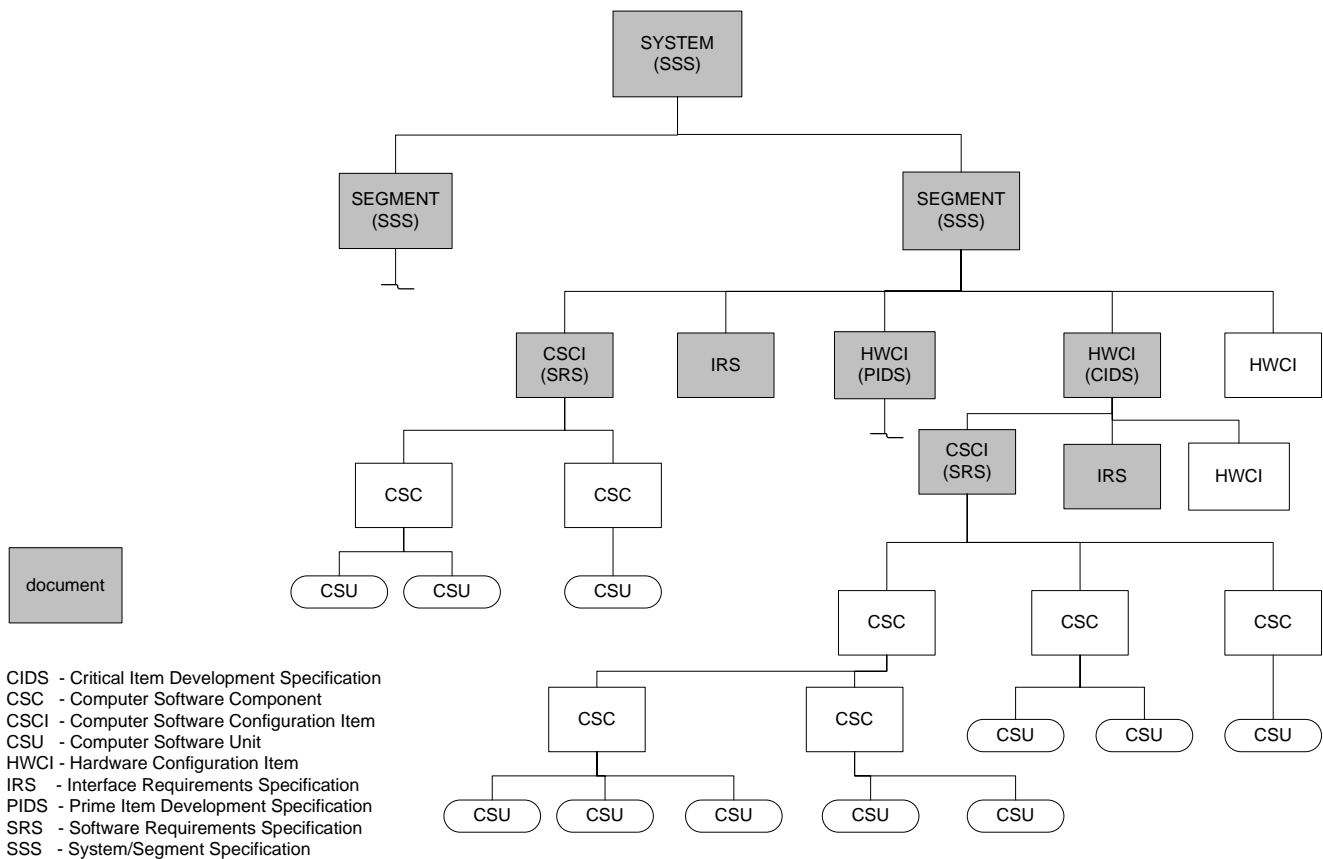


Figure 3 - DOD-STD-2167a System Breakdown

DoD-2167a applies a waterfall like development method to the decomposed system. Figure 4 shows the deliverable artifacts, reviews and baselines that are expected from each phase of the development. There is a separate hardware development path that splits off just after the System Design phase and merges back at System Integration and Test phase.

Establishing Great Software Development Process(es) for Your Organization

In DoD-2167a, any artifact that drives downstream work must have a baseline established. This doesn't mean that the documents can't change it just means there needs to be a 'formal' way for notifying all interested parties that something has changed.

Other factors that add to a project's complexity

Generally, military standards are for very complex projects with huge development teams. DoD-2167a is too formal and rigorous for direct application to small projects. It is useful, however, to understand DoD-2167a as one of the extremes in the range of standards. It is also interesting to explore because, DoD-2167a was one of the last "prescriptive" software development processes. In other words, subsequent process work defined objectives for the software process to achieved, not an explicit process. This difference in process approaches is analogous to creating good requirements. The focus is on 'what' needs to be done vs. 'how' something is to be done.

Embedded software tends to have more constraints than other software projects. Things like microcontroller selection (limited memory size, limited I/O) and hard timing deadlines lead to a need for more up-front design.

Figure 5 adds a couple additional dimensions to a DoD-2167a like process. It not only shows the phases and deliverables, but who is involved in creating the document. It also identifies dependencies with other development items, like first hardware availability.

A Swim lane diagram, also know as a Cross-Functional Flow Chart , is useful for highlighting added complexities which could lead to the need for a more formal process. For instance, more developers and more development groups (hardware, software, systems) leads to more interaction between developers and consequently a more formal process. Dispersed development teams also require additional process rigor.

Think about in your software designs, if an output-driver only has one 'master' that 'master' can interface directly with the driver. However, if there are multiple modules that need to change the output, the driver can't know which request to honor; a manager function needs to be introduced to arbitrate between the multiple requests.

A more complex embedded system (multiple boards or multiple microprocessors) also leads to a need for a more formal process.

Establishing Great Software Development Process(es) for Your Organization

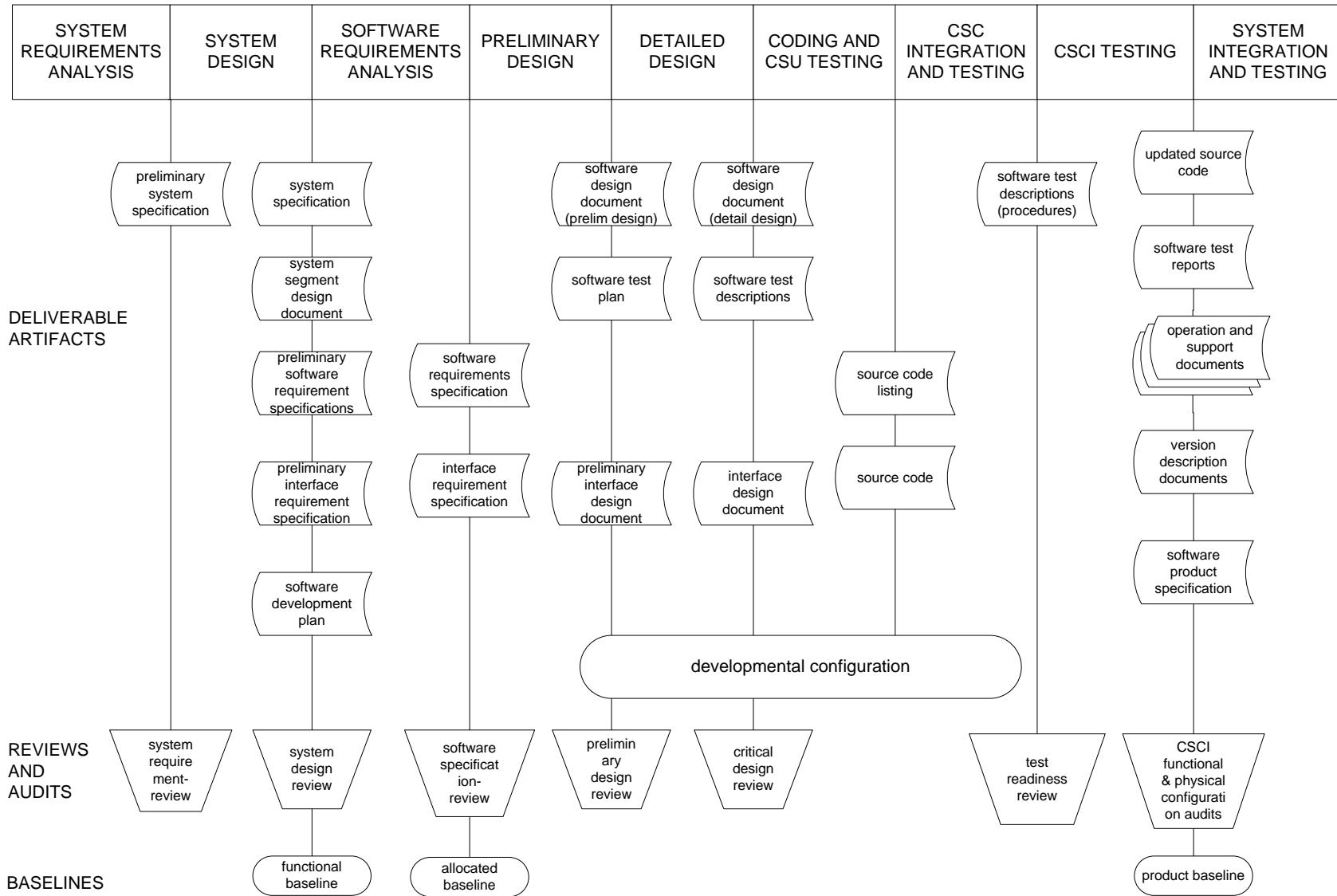


Figure 4 - DOD-STD-2167A, Development Phases, Deliverable Products, Reviews / Audits, and Baselines

Establishing Great Software Development Process(es) for Your Organization

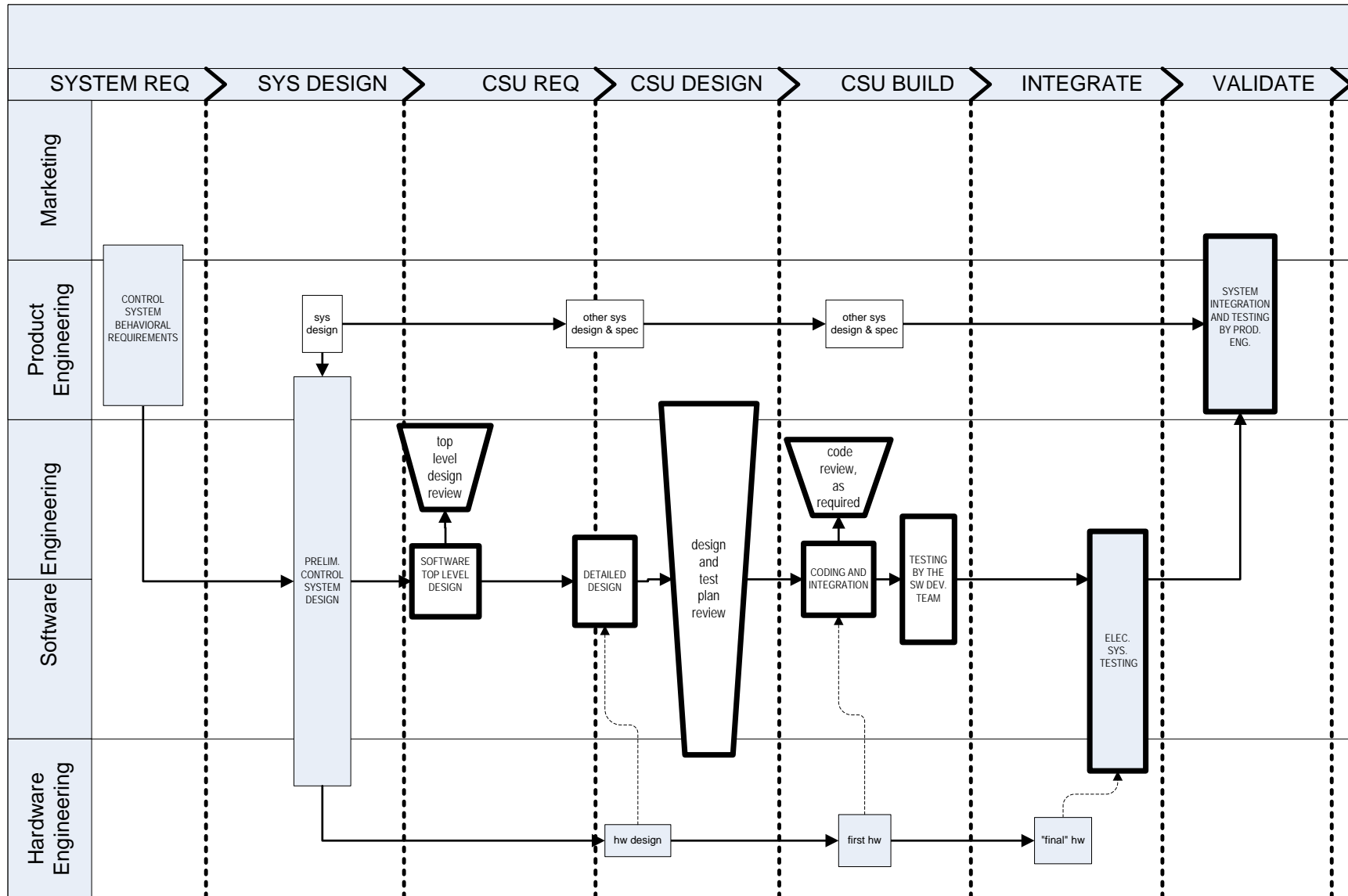


Figure 5, Swim Lane Diagram

Conclusion:

The key points from 2167A that should be applied to a typical embedded software process:

- Do a Top Level Design and Review before starting on the Detailed Designs.
- Do the Detailed Design and Reviews before starting to Code.
- Create test plans and cases in parallel to developing the code.
- Unit Test the code as it is developed.
- Review and Baseline of any document that feeds downstream work.

With an iterative implementation, appropriately functioning code can be available earlier in the development cycle.

Design reviews are very effective for finding errors. They are also a great way for a group to share best practices and train junior developers. Before you can have an effective design review, you need to have a documented design.

Thru this paper and presentation, you should now have an idea of when a more formal process may be needed and what aspects of a formal process make sense to apply in your company.

Contacting the Author:

The author welcomes questions or comments on this paper and can be reached at:

Dale Mayes

Founding Member

Home Port Engineering LLC

20004 103rd Court NE

Bothell WA 98011

Dmayes@HomePortEngineering.com

Phone: (425)876-1915