Where Software Comes From

Last update 31 Oct 2018 [MLS]

[NOTE: As with any compendium of public knowledge this document is under constant revision. Don't be surprised to find something changed from the last time you read it. And don't be hesitant about making improvements as you come across better ideas.]

Product development has to start from a very clear set of requirements and follow a regimented process in order to end up with a predictable result. Here's how.

Clarify Product Requirements

Clear and concise product requirements don't just fall out of the sky into our lap. A good process for identifying and fully resolving requirements will benefit everyone who comes into contact with a product development project.

Requirements may come from any number of sources, but they eventually must be condensed into a single set. To be valid requirements must be written down, inspected, and accepted by everyone responsible for effecting the desired result.

Use a Glossary

(See "Entitiy Naming")

The absolutely quickest way to insure a project's demise is to use terms that have different meaning to different members of the product development team. To combat that you need a glossary of terms so that all team members know what is being discussed. Terms, phrases, acronyms, and abbreviations need to be defined in clear and unambiguous words so that they cannot be misunderstood.

This entire exercise may seem like a waste of time. It is most certainly not. A properly maintained glossary keeps everyone synced up and enlightens those afraid to reveal their ignorance to the rest of the team. In addition, this glossary is useful to people outside the product development team who need information about the project.

Create New Terms

When new concepts come up in discovery the team will have to create new terms to identify these concepts. Use a thesaurus. Try to minimize the verbiage in the new item's description. Use words that accurately describe the nature of the item, but only the facet in which you are interested.

The creation of new terms is an art, so take your time and do it right. A good term can last the lifetime of the product and beyond. A hastily-created term ends up causing confusion and disarray, eventually needing to be replaced by a better term.

Continuously Expand the Glossary

Throughout the discussion of requirements there will be terms that Marketing, Sales, Engineering, or other groups within the product development team will use that are unfamiliar to the other groups. As soon as a new term is introduced into the discussion it should be added to the glossary so that its meaning is plain.

Use One Term Per Concept

Don't reuse terms. Make sure that different concepts are named differently enough that their differences are obvious to the casual observer. If a better term arises to identify a concept, use it. If another concept surfaces that can make better use of a term already deployed, use the existing term for the new concept and find another term for the old concept.

Define the Product Domain

Marketing will say they want a kitchen appliance. Engineering will say they can build a toaster or a tea kettle. One or the other. Marketing will need to decide if they want to toast bread or brew tea.

The domain of the product determines the baseline of behavior of which the product will be capable. Marketing cannot ask for a toaster with a spout. Engineering needs to stick to this principle in order to set reasonable limitations on the changes that will come later.

Adapt to Change

Note that changes *will* come later. This is not a possibility, it is a fact. *Nobody* can come up with all the requirements for a new or altered product from the get-go. Rather than dreading the inevitable discovery and mind-changing that takes place not only in the offices of Marketing professionals but also in the workplace of the customers, Engineering should be prepared to embrace these changes as improvements-in-place of a product that will end up better for them.

Identify the Critical Success Factor

Eventually Marketing will need to select one particular aspect of product operation as the overriding concern. The product has to do *this*, otherwise it isn't a product at all. The toaster has to toast bread, both sides, evenly, and without burning. Without meeting these goals the toaster is a paperweight.

Marketing may need some help in determining what the Critical Success Factor really is. To that end Engineering will need to ask leading questions about what defines success and what is failure.

Constraints and Capabilities

Marketing says they want a toaster. Engineering then asks how many slices of toast they want to cook at once. Marketing says two, although four would be real nice. Engineering tells Marketing what is involved in making a four slice toaster versus a two slicer. How much more expensive the power supply would be, how much more expensive the casting, molding, and other assemblies would be, and how much more development effort would be involved. Of course, the development schedule would stretch out, also. Marketing reluctantly agrees to stick to a two slice model.

Remember, requirements are about *setting constraints* on a product just as much as they are about defining what capabilities a product has. This is because having the constraints in place up front puts limits on the flights of fancy in which all parties tend to engage during discovery and feature expansion.

So, although the Marketing team decides they want a toaster versus a coffee maker, this constraint need not be specified. (After all, there's no place to pour the water into a toaster.) However, some constraints *do* need to be specified, like saying the toaster will do bread, frozen waffles, and those skinny pastry things, but it will not do bagels. Why? Because bagels are thicker than other foodstuffs and the toaster can't handle something of that size. With this constraint in place no further discussion of bagels need ever come up.

Sometimes the slightest change in product requirements can have significant impact on the underlying structure that supports the base functionality. These technical issues are never readily apparent to those outside of the Engineering disciplines, and sometimes not even to those who have skill, training, and experience in developing products. Whenever Marketing wishes to change a feature or a specification it is incumbent upon Engineering to do the due diligence required to determine if this change is significant or not.

By establishing the limits of specification flexibility Engineering can with one shot resolve the problems of shifting goal posts and simultaneously provide boundaries for the daydreams of Marketing free thinkers. This is also how the team as a whole discovers *what they don't know* about the product, and the approximate limits of their ignorance.

Determine Relative Values for Features

Product specifications are limited by any number of factors. These may include cost, size, weight, complexity, time-to-market, manufacturing turnaround, etc. Most product specs are limited by some combination of factors. In the same way, the tradeoffs that Marketing must make in choosing functionality for the product must be made within a set of specified limitations.

The "zero sum game" discipline is particularly suited to establishing the relative values for all functionality that a new product is to have. The idea is to set some target value for the product's sum of features, then have Marketing assign relative values to the features they want. The sum of relative values must exactly equal the target value.

This exercise forces Marketing to examine all of the functionality they have requested as a total package. It is highly likely that Marketing will make changes to the requirements once the relative values of certain features come to light. It is also likely that Engineering will be able to offer advice on what features can be grouped together and treated as a single value due to these features' reliance on a shared component of infrastructure.

Engineering can also point out what feature(s) cause the greatest increase in product cost and complexity through requiring additional support. Just as one invocation of sprintf() will cause the linker to bring in 85% of the C runtime library, one seemingly innocuous feature "enhancement" can force the inclusion of massive electronic and software support in the product's underlayment.

Explore the Alternatives

However, there may still be a way for Engineering to save the day here by suggesting alternative features that end up providing the same functionality. For instance, instead of a complex GUI with a color display, touch screen, and operating system behind it, Engineering may suggest the use of a network interface with remote operation. All of the sophisticated display and control functionality can then be offloaded on to a remote desktop application or even a web browser. The cost and time-to-market for the main product decreases, and the remote application can be developed in parallel and for less cost than the internal GUI. Toss in the fact that this solution provides network functionality "for free" and Engineering has made a sale.

Bear in mind that Marketing still needs the freedom to change requirements within the limitations that Engineering can provide. If Marketing chooses to use an alternative technology or feature set suggested by Engineering then there may be additional changes either to the scope of these features or their scale. For instance, now that the product has a network interface and is capable of remote operation, Marketing may ask for many more internal datum to be made available on the network. Reporting is a very valuable feature in many products, and this is the kind of thing that Engineering can expect to be asked to provide (and, later, expand) once the capability is there.

Finalize the Requirements

Eliminate Waffling – "Put A Number On It"

Requirements need to be definitive. They must be expressed in measurable quanta, not vague qualitative assertions. The requirements document can never contain words like should, could, might, "as possible," or any other non-definitive word or phrase. Even those characteristics that can't be quantified directly can be qualified relatively. Therefore, almost any characteristic of the product that is important enough to justify a mention in the requirements can be expressed as hard numbers if Marketing and Engineering are willing to work together to establish measurements.

Remove Subjective Measurements

For strictly subjective factors Marketing will need to test a prototype or spike solution to settle the question of what is better. For instance, Marketing may require the product be "easier to use" than the old one or the competition. That can be defined by fewer keystrokes or by graphical representations versus numeric displays, etc. "Ease of use" testing may employ off-the-street testers to see if they can use the new box more easily or get work done in less time than the old product.

(Side note: Be aware that there will probably be facets of usage that will be the focus of unending debate. At some point someone in authority will simply have to call an end to research and make a determination of what the requirement will say. Everyone will have to agree to live with the results, even if they are less than perfect. Just as baseball always has the next season, you will always have the next product.)

Marketing may require the product be "higher quality" than the previous product or the competition without really specifying how that quality is to be measured. Engineering then asks questions about mean time between failures, rate of repair calls, hours of tech time to fix field failures, cost of maintenance, etc.

Marketing may require the new product to be "smarter" than the old one or the competition. Engineering then asks about decisions the old product could make on its own and compare that to the decision capability in the new product. If it turns out that the new product still can't make better decisions without user interaction then perhaps the new product needs additional sensors, CPU horsepower, memory, or other features that Marketing didn't realize were needed. These things can be quantified.

Amongst these criteria Marketing will find characteristics they consider meaningful and want those applied to the product. So, the result of all this research and study goes into the requirements as a specific group of features that are related, but also expressed in numbers.

Achieve a Consensus

To achieve consensus on the requirements they need to be put into a formal specification document which can be formally inspected. The goal here is not to set the requirements into stone so that they can never be changed, but to establish what requirements are absolute, which ones are malleable, and identify the difference. As long as all the participants in the creation of the specification document are fully aware of this there will be no shifting goal posts later.

A requirements document can cure a lot of problems long before they arise by crystallizing all of the research and decisions that went into the final set of specifications. Keep in mind that the inspection process lends credence to the result of requirements refinement and produces a trustworthy document as the result. Creating a specification, though, is an art unto itself.

CREATE A SPECIFICATION

Set Constraints

Once the baseline characteristics for a product have been defined the next step is to set hard limits on all the features. Engineering can't just make up numbers for these limits; the entire team will have to work hand in hand to establish specifications that include real numbers and not just fuzzy concepts.

Marketing will instantly complain that this violates the "freedom of expression" that Engineering just granted them back in the feature evaluation stage. Not so.

The establishment of hard numbers is so that Engineering can make important decisions about architecture, design, and implementation without constantly seeking approval from Marketing.

Marketing is free to make changes in scale or scope that can be accommodated by the underlying platform. Some changes, though, will simply be too much for the substrate to support. Setting limits here will prevent the problem of discovering – too late – that the last change was the straw that broke the camel's back.

Publish a Usable Document

An additional benefit of having a fixed set of specifications is that everybody who is involved in the project knows what those limitations are. Sales will not attempt to sell a two slice toaster into a customer application requiring a four slice machine. Manufacturing will know that they need not tool up for four slice templates and fixtures; their current two slice operations will be sufficient for the new product. Purchasing will know that they do not need to look for a new vendor who can provide casting and molding service for a four slice backbone. Operations will know that they don’t have to provide new part numbers and shelf space for four slice parts. Everyone benefits.

The specification identifies all of the operational aspects of the product and attaches hard numbers to everything that can be defined numerically. Once this document has been through inspection it is ready to be used by everyone on the team as the "last" word on what the product will be. Of course, the changeable aspects of the specification are still there, waiting to be exploited…

CREATE AN ARCHITECTURE

Architecture identifies the services needed to support the requirements of the product without specifying how those services will be provided (hardware, software, magic, etc). *By definition* architecture is the base, irreducible, and immutable set of rules on which a system is based.

Base

The toaster heats up bread electrically. Therefore, the architecture calls for electrical components to heat up the bread. There is no provision for gas operation at all; no valves, flue, fans, or any other component of a gas system. Architecture calls out what is at the bottom of everything happening in a system.

Irreducible

The toaster cooks bread (a) on all slices simultaneously, (b) on both sides of each slice simultaneously, and (c) evenly across both surfaces of each slice. These attributes are independently stated, but are not independently optional. All operational rules stated in the architecture must be in place for the product to have a baseline of functionality to support the Critical Success Factor.

Immutable

The toaster toasts bread by electrical heating means, doing one or two slices per cycle, cooking both sides of each slice simultaneously and evenly. Once these rules have been established there can be no later change in them. You can’t make the toaster use gas to cook the bread, or do three slices, or do one side of a slice at a time.

DESIGN A SOLUTION

Design defines the mechanical, electronic, and software support needed by the services called out in the architecture. Proper design assigns the appropriate responsibility to the components best suited to fulfill those responsibilities. Typical solutions put less emphasis on the mechanical, electromechanical, and electronics aspects of a design and more on the software, the most easily changed component of any system. However, there is no substitute for a valve or a lock. Decisions about when to turn the valve or open the lock can be postponed to software.

Further Use of Critical Success Factor Analysis

Requirements gathering is not the only place where Critical Success Factor analysis can play a role. This same thinking can be applied to any component of a design. The question should always be, "What is the base result of this operation?" By answering that question at every separation of components Engineering can fix targets for the design that can be measured and fulfilled definitively. A heating element must produce heat. If it doesn't do that then it isn't a useful component regardless of its safety, cost, ease of replacement, manufacturability, or any other consideration.

Postpone Decisions

Decisions that are fixed in mechanical, electromechanical, or electronics hardware can't be easily changed later in the design process. Decisions that can be postponed to software are easy to change so as to accommodate unforeseen circumstances that arise during discovery. Therefore, the design should almost always feature a minimalistic approach to tasking the hardware, with the main focus of attention being devoted to software.

Achieve Balance in All Things, My Son

Figuring out how much tasking to devote to electromechanicals versus electronics hardware versus software is always a balancing act. For some aspects of operation it's easy – fluids have to be controlled with a valve, period. So you know you need a valve someplace.

The next tricky question is how to operate that valve? Stepper motor? Solenoid? DC motor drive? Then, how do you process the valve motion? Is it a simple open/close? Linear movement with flow rate compensation? Do you have to apply a PID motion control system to the valve movement?

Costs usually play an important role in determining how much effort is assigned to each level of a system's solution. However, looking ahead Engineering may see a lower overall solution cost by adding capability at various levels to accommodate reasonable and foreseeable changes in the product's requirements. Of course, if Marketing won't go for any additional costs then Engineering needs to be firm in stating that the most immediately cost effective solution is also a limited one. Engineering needs to specify the limits to what a particular design can do versus what flexibility a certain amount of additional cost will provide. Write all this stuff down for future reference, because the topic may come up a time or two. *<cough, ahem>*

Complete the Low Level Design in File and Function Document Blocks

(See, "Software Coding Guidelines" for a more extensive discussion.)

For each component identify its interface elements and specify the data it needs to do its work. Describe the internal functions of the component, the algorithms involved, and the internal data being exchanged in the document blocks for the files and functions that make up the component.

If there is insufficient detail in the design to allow such documentation to take place then the design needs to be refined to address additional detail at the lower level. This process is iterative and not necessarily entirely predictive. This is acceptable; it is rare that a planned implementation will follow its original path without any mid-course corrections during discovery, testing, feature tuning, etc.

The use of file and function document blocks to provide the lowest level of design documentation is the best compromise between having insufficient design documentation and having too much inertia and rigidity in the design. It is up to the skill, experience, and good sense of the design/coding team to determine the proper cutoff point between design and coding, just as it was between the architecture and design. As with any layered plan, the design should identify more of the what (higher level) and code more of the how (lower level).

The actual code must closely follow the algorithm or pseudo-code logic expressed in the file or function document block. If the code needs to change then the document block must first be updated to reflect the changes. It is not at all hard to keep these miniature "documents" up to date, since they are very terse and are co-located with the code in question.

Note that this is *not* the same as "self-documenting code" or any such silliness. The miniature document describes only what needs to be made plain when a cursory examination of the code itself can't do so. This description is created apart from and ahead of the actual composition of the code. It is a part of planning, vital to a successful coding endeavor.

Don’t get too hung up on the exact layout or format of a file or function document block. The idea is just to capture the important information in the most concise and easily-maintained form that can be had. Forcing compliance with some particular format or template leads to mostly empty and useless document blocks that not only don’t convey anything but also don’t get maintained. This is the sure way to kill any progress your team has made towards tracking the low level design.

FULFILL THE DESIGN WITH HARDWARE AND CODE

[Note that this section is still being expanded. Feel free to add value here, folks. We’re all benefitting from this exercise, eh?]

- Design specifies how the services identified in the architecture are fulfilled. At design time the development crew determines the best combination of hardware and software to create these services. Once services are put into place they can be tested and used as standalone packages.
- Hardware fulfills specific tasks with the lowest cost and at the greatest performance, but is the least flexible solution.
- Code puts design into practice by providing a specific set of instructions to a specific set of hardware.
- With most modern projects the hardware is based on previous products or on other well-known platforms. Some of the device drivers and other hardware-level software can be essentially picked up from the hardware’s legacy.
- New hardware will require new drivers, of course. Some of these are picked up from the OEM, but many times the driver will need to be created from scratch or heavily modified to meet a specific BSP or OS limitation. Note that re-use is a wonderful thing, but isn’t a really useful concept in most embedded applications because the hardware platforms vary so much -- even in the same product family. After all, the reason you are making a new product is because the old one can’t handle the demands placed on it.
- Application logic is the last thing to be developed because it is the entity that will change the most. Applications should mostly consist of calls to supporting services provided by the substrate coupled with decisions based on the current conditions and commands from the user.