Student Specifications Document Template Guide

Name1

Name2

Name3

October 6, 2016

Acknowledgements

This template was developed and compiled from multiple sources. It owes its origins most firmly to a mil-spec template that I found on line years ago. That template contained several humorous lines about how boring specifications are and that cleanliness is next to Godliness. I had never laughed before when reading or writing a specification. I liked the author's approach.

The source of that original template is lost but the framework reappeared several years later as a Small Satellite requirements template but I am sure that it was a modification of the original. Next, this template also owes much of its structure to the brilliant book by Phillip Koopman 'Better Embedded Software' a book that describes a development process that is useful for a whole lot more than software.

Much of the current structure and helpful comments are the work of my colleague Jolynne Barrett. She has made the design process much less intimidating for Senior Design students. They tend to lose that 'deer-in-headlights' look after they realize that Jolynne will help them bear the heavy load of project documentation.

Preface

Requirements documents spring out of military/aerospace methods and are part of a formal process intended to minimize needed redesign. These systems design methods are heavily front-loaded requiring extensive pre-design effort. When properly exercised these methods can, and do, produce safe, effective systems that are on time and budget. Requirements documents help with the so-called concurrent engineering process in that all aspects of the project are specified at the beginning of the project down to and including the kind of box that the system is going to be put in. Concurrent engineering lessens the chances of unpleasant surprises late in the program. Hearing things like, "What do you mean it has to fit in a two inch cube?", six months into a program is frustrating and expensive. There are less formal methodologies that demand less up front work, but while it is easy to relax the formal it is often hard to go the other way.

Software developers often advocate design through things Extreme Programming or the Agile methods. Some pieces of these methods are quite useful, e.g. users stories, but as a crusty old design engineer I do not like an approach that uses near constant redesign as a design methodology. When push comes to shove and lives are at stake, engineers will almost always choose a formal systems design methodology.

Many people say that this formal style of engineering development is passé and that newer, speedier, more flexible methods should be taught and used. And yet they are unknowingly depending on these formal techniques every time they step onto an airplane. To those who pine for the less formal methods I always ask the following question:

> *Would you fly on an airplane designed by the methods you advocate?*

The answer is seldom yes.

How to Use This Document

This document is divided into five major sections. While specification document formats vary from group to group the format presented here is a good representation of what you will encounter in industrial specifications. The sections of the document are:

1. *Scope*
2. *Applicable Documents*
3. *Stakeholder Requirements*
4. *Engineering Requirements*
5. *Verification of Requirements*

This document is not only intended to explain engineering specification documents and show the development of a specification through the use of a running example. It also aims to help you produce a specification for your project. In order to achieve this goal each section will be presented in three levels. I will:

1. *Attempt to explain the reasons that the section exists and what data goes into that section.*
2. *Complete that section for the running example and annotate the example to make it more understandable.*
3. *Strongly suggest that you complete the section for your project. When you see the symbol*  *it means you are going to get to produce something.*

A skeleton template will be available in the same place that you found this document. Fill out that skeleton template as you follow through this document. You will quickly produce a specification in the least painful way I can think of. Note that I didn't say pain free. Sometimes thinking as hard as you need to think in engineering design just makes your head hurt.

Signature Page

Signature	*Signature*	*Signature*
Date and email	*Date and email*	*Date and email*

Revision History

Revision	Description	Author	Date	Approval
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

A simple hardware example will be used throughout this guide to illustrate the writing of a specification document. The USU ECE Controls Lab needs a new power amplifier. The requirements for its design follow the narrative in this document.

From experience I know that I will not get it all right the first time. If you find what you believe to be a mistake or omission, and can make a good case for why you feel that this is so, then I will revise the document and immortalize your name in the revision table.

Contents

1 SCOPE **9**

2 APPLICABLE DOCUMENTS **10**

3 STAKEHOLDER REQUIREMENTS **12**

3.1 Stakeholders User Stories . . . 13

4 ENGINEERING REQUIREMENTS **19**

4.1 Interface . . . 26

4.1.1 Functional Interface Constraints . . . 26

4.1.2 Functional Interface Requirements . . . 26

4.1.3 Support Interface Constraints . . . 26

4.1.4 Support Interface Requirements . . . 26

4.2 Functional Requirements . . . 26

4.2.1 Functional Method Constraints . . . 26

4.2.2 Functional Design Requirements . . . 26

4.3 Support Requirements . . . 26

4.3.1 Support Method Constraints . . . 26

4.3.2 Support Requirements . . . 26

5 VERIFICATION OF REQUIREMENTS **27**

5.1 Verify Coverage of Stakeholder Requirements . . . 28

5.2 Interface . . . 29

5.2.1 Functional Interface Constraints . . . 29

5.2.2 Functional Interface Requirements . . . 29

5.2.3 Support Interface Constraints . . . 29

5.2.4 Support Interface Requirements . . . 29

5.3 Functional Requirements . . . 29

5.3.1 Functional Method Constraints . . . 29

5.3.2 Functional Design Requirements . . . 29

5.4 Support Requirements . . . 29

5.4.1 Support Method Constraints . . . 29

5.4.2 Support Requirements . . . 29

Specifications

1 SCOPE

This section is a very top level, simple verbal description of what the item is and where it is used. See the amplifier example under the **General** *heading.*

(a) **General: This document describes the design and verification requirements for the USU Controls Laboratory power amplifier module. The module is used to provide current gain to the control voltages that drive the laboratory motors.**

For a simple module such as the power amplifier module the above description is adequate.

(b) **Acronyms:** *I have not chosen to define acronyms because I have a thing about the overuse of acronyms. You may not be so burdened and find that typing out a name is simply too tedious. Put those acronyms here.*

(c) *Additional short descriptive paragraphs can be added only if needed for special classification, designation of alternate versions or other material that is part of a top-level description.*

 *Take time to write this section for your project. Being able to write a simple description of even complex things is a good indicator of how well you understand what you are planning.*

2 APPLICABLE DOCUMENTS

This section is often poorly understood and poorly implemented. It is a list of documents that are a critical part of understanding the item or requirements imposed on the item. Every document listed must have a text reference in the body of the spec further describing and limiting how it is to be applied. Conversely, no document is to be referenced in the spec unless it is listed here. Don't put items here that are background information or of general interest. Always obtain and review all items listed here. This section often has the following statement:

> *The following documents shown shall form part of the specifications for this project. In the event of a conflict between requirements, priority shall first go to the contract, second to this document, and lastly to these reference documents.*

There are lots of MIL-STD(standards) and MIL-HDBK(handbooks) that cover an amazing range of subjects. Here is a website that has them plus NASA documents and others all available (every government specification and handbook you can imagine) for free. Other groups publishing standards include Institute of Electrical and Electronics Engineers (IEEE), American Society of Mechanical Engineers (ASME), Society of Automotive Engineers (SAE), American National Standards Institute (ANSI), American Society for Testing and Materials (ASTM) and Aeronautical Radio, Incorporated (ARINC) to name more than a few.

Another thing to note is that referencing documents will often save you time (and money). For example, a referenced document can contain a complete set of environmental tests. Stating that your system has to be tested to the requirements of MIL-STD xxx is a lot easier than making up the series of tests yourself. The process is akin to what you do when you use a library in C programming: someone has provided software that meets your needs. Why reinvent the wheel? Particularly in something as difficult to get right as environmental testing.

Another benefit to using standards documents is that it connects your project to what is typically done in industry. While this should be obvious it is sometimes forgotten. The designers that you hand the spec over to may already know the standard that you have specified and know how to meet these standards. The simple adherence to standard could save you a lot of money.

(a) **Government Documents** *This is where to put MIL-Specs, MIL-STDs, NASA specs and so forth. Be sure to include the revision level and date.*

(b) **Industry Documents** *This is where to put ANSI, ASTM, ASME, IEEE, Company specifications and so forth. Both this section and government documents can be divided up into logical subcategories.*

My project is pretty simple and I don't have any applicable documents, but your project might. Did you know that in order to use USB on a commercial device that you have to pay a licensing fee? I know, your development board has a USB port on it. You can use it because the board manufacturer paid the fee. The USB spec is about 1500 pages. Saying that your device must comply with this spec (or a part of it) is a lot easier than writing it all out.

 *Research and write down the external specifications that you want your system to meet. Think about standards (like USB) and MIL-Specs that cover things like environmental testing and electromagnetic compatibility.*

3 STAKEHOLDER REQUIREMENTS

This section is fundamentally a list of characteristics and features requested by the stakeholders. Stakeholders are anyone that has part in the system whether in designing it, packaging it, using it, maintaining it, etc. Forgetting or ignoring a stakeholder often means change at a project phase when change is expensive.

As an example, you are tasked to design a large temperature display for a bank. The typical 'customer' only passively interacts with the device through its display therefore having no input to the system. However, the designers and maintainers of the system must have the ability to modify the system and evaluate their modifications through the display. These users access the system both through input and output and their needs must be met by the design.

I have identified the stakeholders in the power amplifier module project listed and they are listed in the specification as follows.

The stakeholders for the USU ECE Controls Lab Power Amplifier Module are:

1. The course instructor
2. The lab instructor
3. The lab student
4. The USU ECE Department

 *Think about the stakeholders in your project. Write down a list of stakeholders in your project on your specifications template. Think about your project as if it were to be produced commercially. Think deeply.*

Gathering Stakeholder Requirements

One approach to writing client requirements is to use what are called 'user stories'. User stories are essentially the information that a user (stakeholder) will give you when asked what the system must do for them. See the following website for more information on user stories:

(User Stories).

Some users are more difficult to get stories out of than others as this Dilbert cartoon attests.

(Alice Tries to Extract Requirements From a Stakeholder).

Requirements answer the 'what' questions about your project.

- *What must it do?*
- *What must it weigh?*
- *What environments must it endure?*
- *What must be done to ensure safety, maintainability, cost, etc.?*
- *What constrains the design (and designers)?*
- *+ many other potential questions in many categories.*

It is important that you don't try and complete the design at this step. We don't design (specify the 'how's) at this point for two primary reasons: we may be only specifying the project and then turning the requirements over to a designer or we don't want to pigeon-hole our thinking this early in the design process. Keeping an open mind at this stage of the design often leads us to better thought-out designs. You should marginally note your ideas of how to meet the requirement, but do not require a specific 'how' unless it is truly a design constraint. Don't worry, there will be plenty of time to figure out how we are going to meet the requirements of the project a little later in the design process.

Requirements typically divide themselves into design constraints, functional requirements, and non-functional requirements. For example, a functional requirement would state that the device shall measure and record barometric pressure with such and such accuracy every minute over a 24 hour period. A non-functional requirement would require that the device operates on no more than 500mW average power with power maximums of 1W.

We see an example of a true design constraint in the WISE scientific instrument designed by the Space Dynamics Laboratory at Utah State University. The systems designers required a specific FPGA instead of a designer chosen micro-processor simply because they knew that there wasn't any micro that would function in the expected space environment.

Continuing with the power amplifier module as an example the stakeholder user stories are:

3.1 Stakeholders User Stories

The primary stakeholders needs are described below.

- **The course instructor is the primary technical specifier of the controls lab power amplifier. The device:**

1. Must meet the pedagogical requirements of the course (ECE/MAE 5310) for which it is designed.

   User Story
   As a course instructor I need equipment in the control systems lab that provides students with a useful hands-on experience.

   In order to be useful the lab equipment used must not be mysterious (or overly 'black box').

   Students need to see the inputs and outputs of the system and understand that they can replicate similar systems in their careers.

   The power amplifier module must be constructed of a technology that is familiar to both mechanical and electrical/computer engineering students at a late-junior year level. In order to meet this familiarity requirement the design is constrained to use a power operational amplifier.

   The power amplifier needs to be a unity voltage gain non-inverting buffer.

   The power amplifier must provide a minimum continuous output current of 8 amps.

   The input impedance of the power amplifier must be greater than or equal to 10 kilohms.

   The largest time constant of the power amplifier must be less than 0.001 seconds.

2. Must not present safety/shock hazards to any user.

   User Story
   As the course instructor, I need a safe module for the students to use. Safety implies freedom from both electrical and mechanical hazards.

   The module must be free of sharp edges.

   The module must not present a dangerous shocking hazard to students.

   In order to eliminate a potential dangerous shocking hazard the power amplifier module supply voltages are constrained to: $V_+ \leq 18$ volts and $V_- \geq -18$ volts or a total supply swing of ≤ 36 Volts.

3. Must be robust so that student wiring errors do not damage the module.

   User Story
   As a course instructor I have observed that mis-wiring is the most frequent cause of electronic failure in the lab.

   The module must detect power supply mis-wiring where grounds and/or V_- and V_+ are interchanged with ground or each other.

   The module must be designed so that such wiring mistakes do not harm the module.

   Additionally, the module needs to inform the user that it is miswired.

   Additionally, the module must protect itself against amplifier output shorts to ground or other over-current conditions.

4. Must be packaged to meet a specified envelope.

   User Story
   As the course instructor I need a simple compatible mounting for the power amplifier module.

   The controls lab uses a one inch t-slot aluminum extrusion railing to mount motors, controllers, and sensors for laboratory experiments. The power amplifier module must be mounted on the t-slot rail.

5. Must have a simple to understand interface that minimizes wiring mistakes.

   User Story
   As the course instructor I need a module that has a voltage input, a voltage output, and power supply inputs. The input and output voltages can, but are not required to, have a common ground.
   Color coded banana jacks are recommended for the module wiring. Specialized connectors are not allowed due because their use separates the student from the function.

- **The lab instructor is the primary maintainer of the device. The device:**

1. Must be robust.

   User Story
   As a lab instructor I am tasked with overseeing and assisting six lab groups of three students concurrently. Fragile systems that do not tolerate student mistakes frustrate both myself and the students.

   Since I am unable to observe all student groups at the same time and since the students will not, in general, wait for me to get them started I need a system that tolerates students mis-wiring.

2. Must be easy to setup.

   User Story
   The more complex the setup the more prone to error.

   Students can follow a color code and this helps then setup quickly and minimizes mistakes.

3. Must be easily repairable.

   User Story
   In the heat of battle, students can be less than patient in waiting for a repaired or replaced module.

   I understand their impatience, but it helps to have easily repairable or inexpensive replacement modules to mitigate the students' impatience.

 *You are a student so you can probably identify the student's user story for the power amplifier module. Take a break from reading this printed morphine and write down a user story for the typical harried lab student. If you get one that I missed you may be able to convince me to revise the document.*

- **The lab student is the primary user of the device. The device:**

    1. Must be easy to setup.

        User Story
        As a student, I want my instructors to be aware that I am busy and that this course is not the only one that I have. Labs that require long or complex setups are seldom worth my time. I want a setup that I learn from and that increases my understanding of what I am studying. Color coding helps me avoid wiring errors and speeds the setup.

    2. Must be easy to functionally understand.

        User Story
        Good labeling helps me to understand the function of the module so that I can compare it to what I have learned in class.

    3. Must be robust.

        User Story
        I am a busy student. Parts that fail easily because I made a simple mistake are frustrating.

- **The USU ECE Department is funding the project. The device:**

    1. Must be low in initial cost (design and prototype).

        User Story
        The department would like to replace the existing power amplifier modules in the controls lab with something that is much less expensive. The current modules from Quanser cost too much.

        The replacement module must be a simple inexpensive design that will not be too expensive to design and prototype.

2. Must be low in replacement/repair costs.

   User Story
   In quantities of 10 or greater the completed module should cost less than $100.

3. Must meet the pedagogical requirements of the course (ECE/MAE 5130) for which it is designed.

   User Story
   The course instructor sets the pedagogical requirements for this module.

As useful as user stories and user interviews are they are not perfect and your access to users may be incomplete. It is up to you to fill in the missing spots. Some of the areas that are typically missed include power requirements (batteries, wall power, hamsters, etc.), packaging size and weight requirements (standard packaging, wear on your wrist, tow in trailer, or my personal favorite: the portal data acquisition system at Boeing where portable meant you could move it with a fork lift), and operating environments (we live a desert and when we design we often forget humid operating environments).

 *Time to identify and then interview your stakeholders! I like this step. It is fun to hear how others who have an interest in this system view it. There is a classic drawing about how different stakeholders see an airplane. You can see the drawing here (How the Stakeholders See the Same Airplane).*

I realize that as a student you may not have access to all of the stakeholders for your project. You may have to role play with your project partners or an available, yet naive, friend. Pretend is fun, at least you used to like it.

4 ENGINEERING REQUIREMENTS

This is the meat of the document. Remember that it is "requirements" and not methods. The definition here should leave no doubt about what is needed. Engineering requirements differ from stakeholder requirements in measurability, detail, and unambiguity.

Measurability is a key attribute; without it we have no way of knowing that we have built what we said that we would build. In the aerospace industry the process of determining that what we have built meets the plan to which we were building is called verification. Verification answers the question, "Did we build what we said we would build?"

Ideally, detail in engineering requirements means that the designer can set off designing and building the system with little or no extra input from the stakeholders. Since we do not live in an ideal world additional stakeholder input will be required. When additional input is sought the specification should be revised to reflect clarification or change. Reality is why we include a Revision History Block at the beginning of the specification.

An ambiguous requirement, by its very nature, means that I could design and build the system in one, two, or many 'wrong' ways. Since the stakeholder who is paying for this system doesn't want one of the many possible wrong ways we need to ensure that requirements cannot be misinterpreted by the designer.

Understand:

1. *"An engineering requirement is a statement about the system that is unambiguous. There's only one way it can be interpreted, and the idea is expressed clearly so all of the stakeholders understand it."*
2. *"An engineering requirement is binding. The customer is willing to pay for it, and will not accept the system without it."*
3. *"An engineering requirement is testable. It's possible to show that the system either does or does not comply with the statement." (Steve Tockey)*

Additionally, requirements must be stated in one place only in this document. You can reference the requirement in another part of the document by using its paragraph number. It is often tempting to restate a requirement in another requirement to improve clarity. However, a requirement stated in two places may only get changed in one place during a revision and possibly leading to confusion or worse a flawed product.

Engineering requirements constitute contacts between between clients and design teams. Requirements are binding.

Requirements form the base that we design on. A well written requirement keeps us from ending up with a less than ideal implementation. Defining requirements forces us to think prior to design.

The engineer's job at this point is to make the stakeholder requirements of the previous section into measurable, detailed, and unambiguous. Sometimes this process requires that a stakeholder be interviewed again until a proper set of engineering requirements emerge.

An old joke in aerospace says that a specification is written and thrown over a high wall to the design group. The spec. writer then runs away from the wall (laughing). While formal adoption of this absurd method leads to disaster, we should write specifications as if we could never clarify anything. The clearer we make the specification now, the lower the cost of change later.

Processing Stakeholder Requirements into Engineering Requirements

Stakeholder requirements in the form of user stories are typically poorly organized and indefinite. A designer cannot design to an indefinite requirement. How would a designer know if an indefinite requirement was met? Also, different stakeholders may have requirements that effect overlapping areas of the system but because of the nature of story collection the overlap in these requirements may not be obvious. There are many more issues and a systematic approach to this processing problem should help.

The work of sorting and firming up requirements can be considered tedious. Engineers are famous for hating tedious work. It is why we are constantly automating things that we don't want to do. And when we encounter a tedious task we have a tendency to create a process to limit the amount of tedium that we must endure. Tedious work reduced to a process becomes less tedious if the process is well designed. In order to translate the scattered and indefinite stakeholder requirement into organized and definite engineering requirements we are going to follow a two step process:

1. *Identify and sort stakeholder requirements into fixed categories.*
2. *Rewrite the sorted requirements into definitive engineering requirements.*

The details of the process are explained in the following sections.

Identifying and Sorting the Stakeholder Requirements

At this point in the process we are faced with the task of identifying and sorting the requirements found in the user stories. Finding the requirements is pretty easy. The user wants

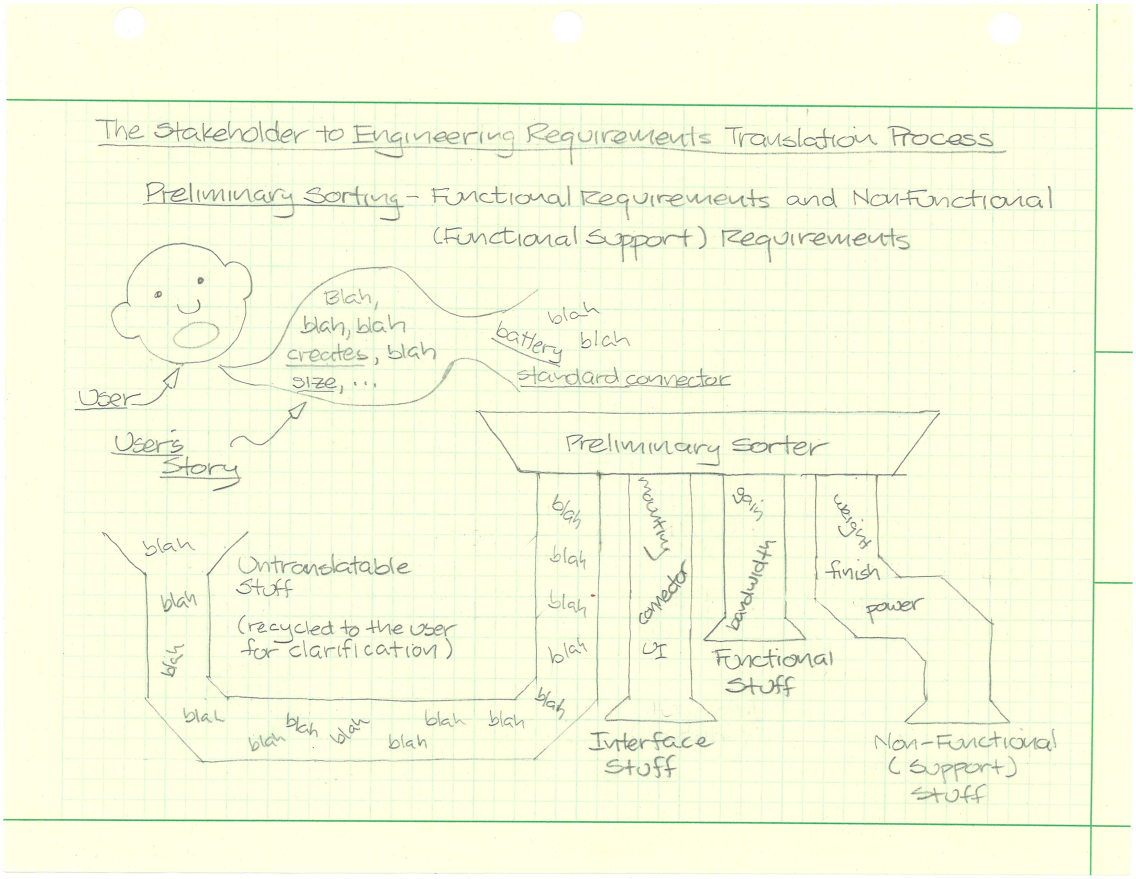

Figure 1: A machine that sorts user stories into appropriate categories.

us to do something and that something is a requirement. Sorting the requirements is only slightly more difficult because we need bins for the different categories of requirements. Different companies come up with different bins so I don't feel too bad at coming up with my own set of bins.

It helps me to think of the process of translating stakeholder user stories into engineering requirements as a machine with several stages. We take the often less than definite requirements that we find in user stories, clarify the untranslatable stuff, and sort the results into applicable categories. The first stage of the process is illustrated in Figure 1.

We will be sorting the requirements into four categories:

- *Untranslatable Stuff*
- *Interface Stuff*
- *Functional Stuff*
- *Non-Functional (Support) Stuff*

The categories themselves are intended to cover all aspects of the project in a systematic and an understandable fashion. Properly chosen categories allow the designer working from this specification to verbally see the system (don't worry, we will also draw a few actual pictures of the system). I'll explain the categories in the next sections.

Untranslatable Stuff

Let's face it we're human and our project stakeholders are human. Being human we don't always communicate what is either meaningful or useful. Sometimes, as engineers, when we hear or read a stakeholder statement it may seem vague or more like a wish. Sometimes the statement may be incredibly obvious, but hard to translate into a measurable requirement. When we encounter such statements it is up to us to approach the individual stakeholder and ask (politely) what on earth the statement means in terms of the project deliverables.

Interacting with our stakeholders early in a project is a good thing. We establish the lines of communication that we will most definitely need sometime in the project.

Interface Stuff

We design and build things to perform some useful or interesting function. Interesting things interact or interface with their environment to send information, provide motive power, receive and process information, receive power, fit into an existing slot, etc. Interfaces make up the useful pieces of a system and are critical to the system.

The documentation for many complex systems includes a very useful document called an Interface Control Document, or ICD for short. This document contains all the interface information for the system. It is a very useful document and helps designers avoid a host of errors. If such a document is available when the specification document is being written then it is simply referenced in the Applicable Documents Section of the specification. For our purposes we will define the interface requirements as part of this specification document.

Functional Stuff

The devices we make are intended to do something. The something that they are supposed to do is defined as functional constraints (designer's hands tied) and functional requirements (designer is free to choose the method).

Non-Functional (Support) Stuff

The engineered system is designed to produce certain things and interface in certain ways. To accomplish its function the system needs a lot of support. For example, a system is designed to process input data in a specific digital format and output the processed data in a specific digital format. The format constitute interface constraints, but the connector for the digital formats constitute non-functional or functional support interface constraints.

The box that keeps the system from getting wet constitutes a non-functional or functional support requirement while the data processing requirements constitute a functional requirement. Meeting non-functional requirements is as necessary to the system as meeting functional requirements.

Let's return to the example and start sorting the stakeholder user stories into the four categories. The stories will be highlighted using four colors with the following meaning:

1. Untranslatable Stuff
2. Interface Stuff
3. Requirements Stuff
4. Non-Functional Stuff

Beginning with the Course Instructor's Pedagogical Requirements. The marked version is:

As a course instructor I need equipment in the control systems lab that provides students with a useful hands-on experience.

In order to be useful the lab equipment used must not be mysterious (or overly 'black box').

Students need to see the inputs and outputs of the system and understand that they can replicate similar systems in their careers.

The power amplifier module must be constructed of a technology that is familiar to both mechanical and electrical/computer engineering students at a late-junior year level. In order to meet this familiarity requirement the design is constrained to use a power operational amplifier.

The power amplifier needs to be a unity voltage gain non-inverting buffer.

The power amplifier must provide a minimum continuous output current of 8 amps.

The input impedance of the power amplifier must be greater than or equal to 10 kilohms.

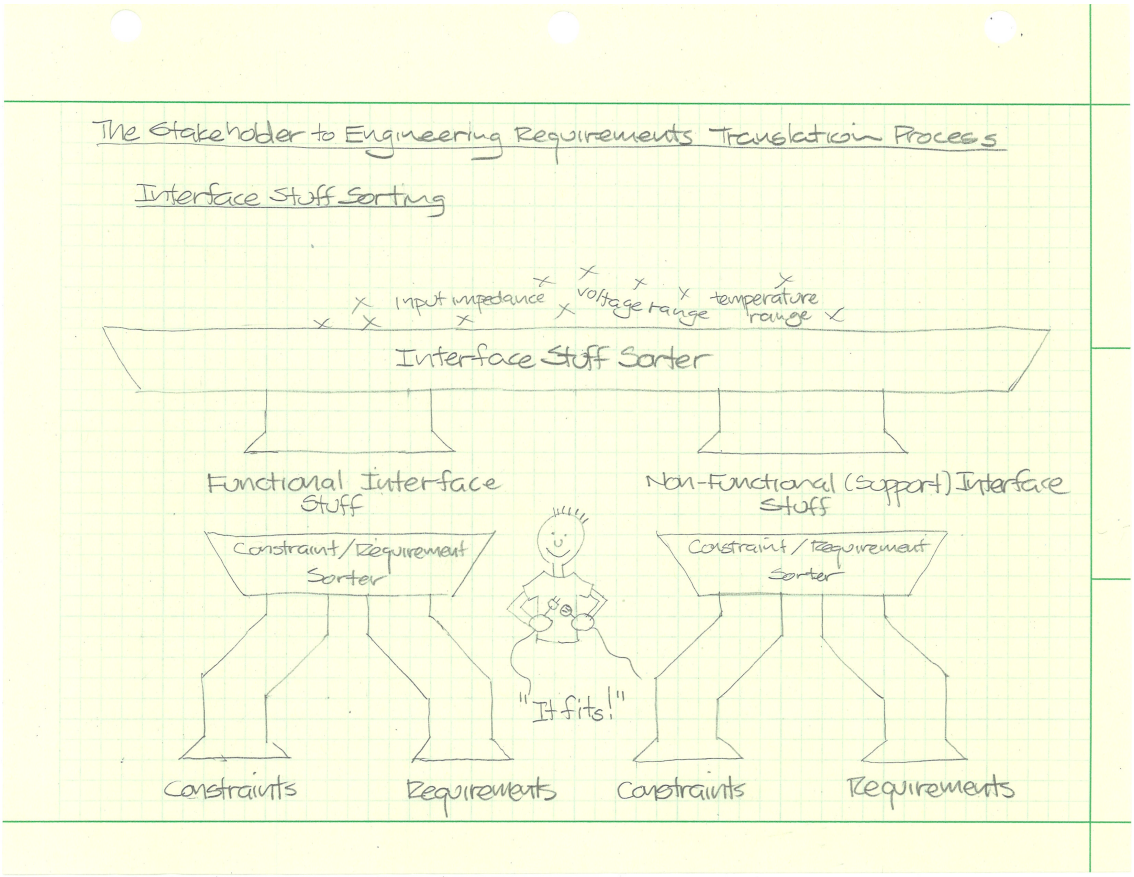

Figure 2: A machine that sorts Interface Stuff into appropriate categories.

The largest time constant of the power amplifier must be less than 0.001 seconds.

The Course Instructor's Pedagogical Requirements has one item classified as Interface Stuff dealing with the input impedance of the amplifier. This item must be further sorted by the machine in Figure 2. This machine determines if the item is a constraint or a requirement.

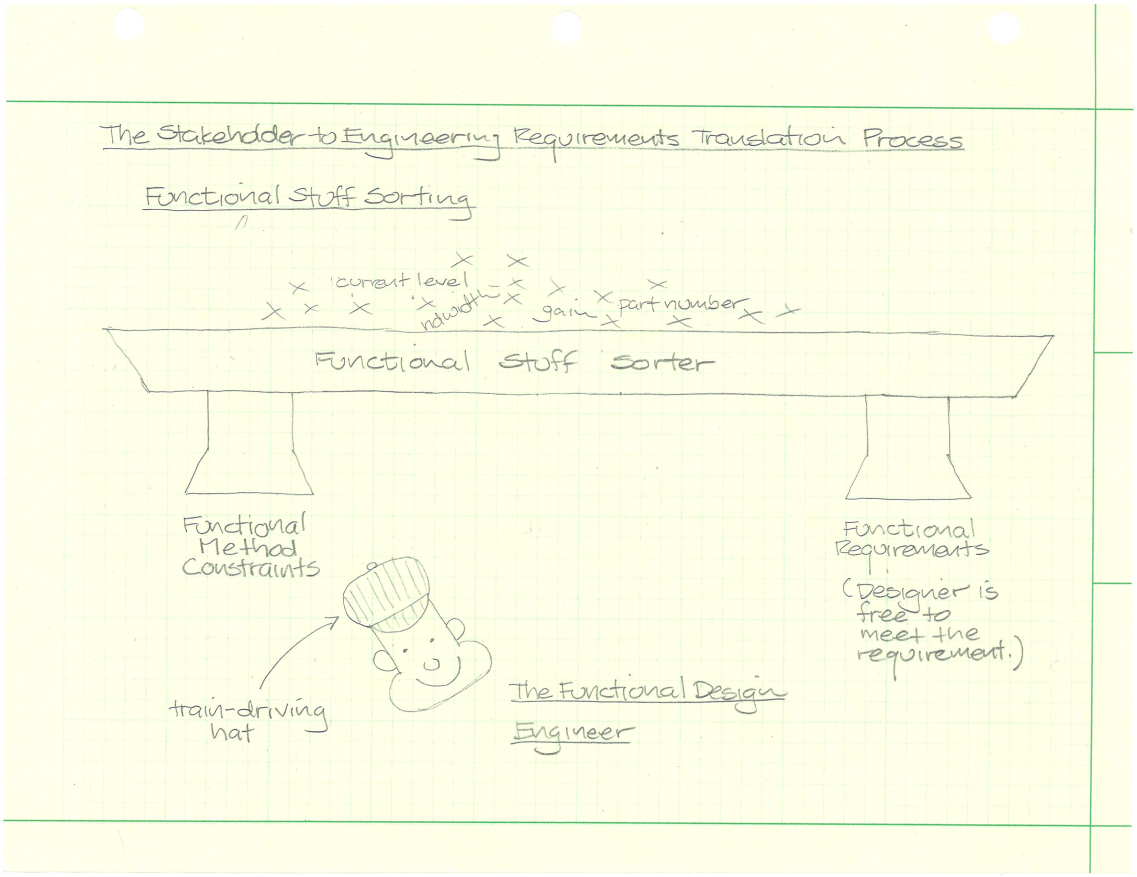

Figure 3: A machine that sorts Functional Stuff into appropriate categories.

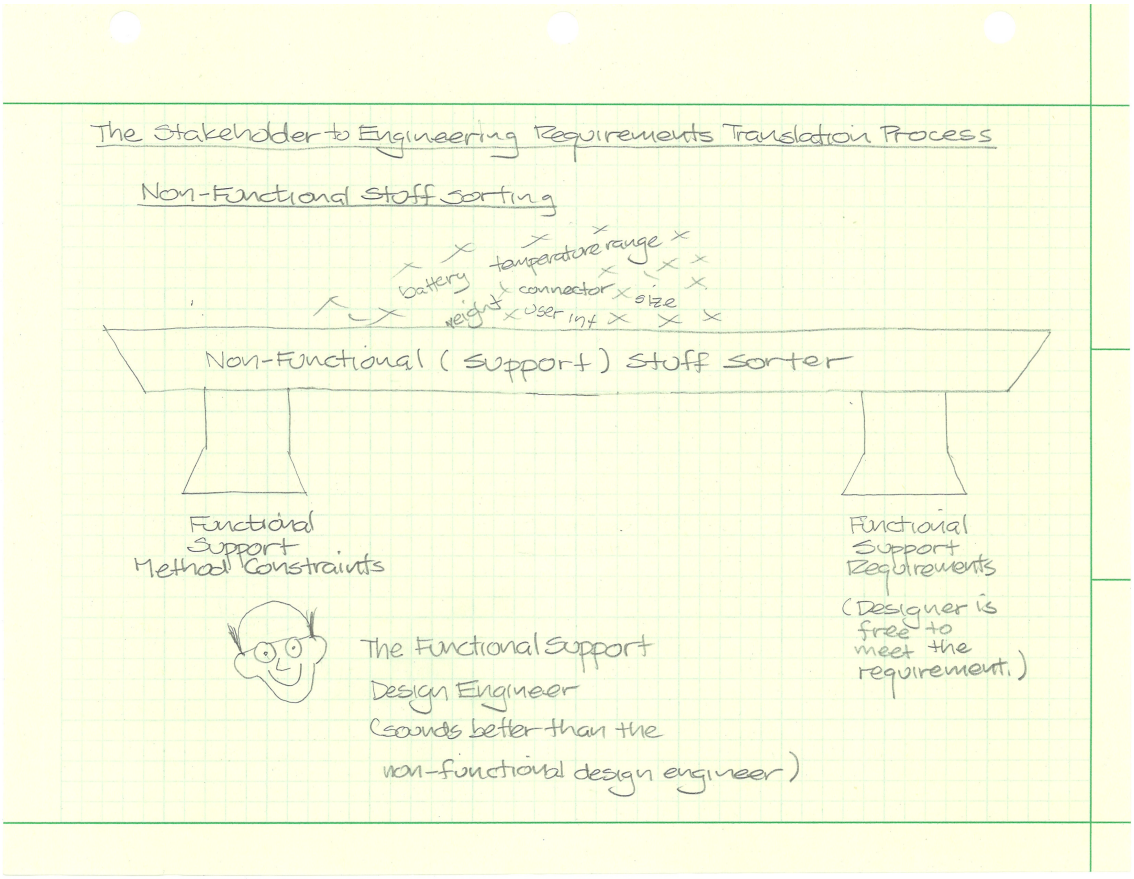

Figure 4: A machine that sorts Non-Functional Stuff into appropriate categories.

4.1 Interface

4.1.1 Functional Interface Constraints

4.1.2 Functional Interface Requirements

4.1.3 Support Interface Constraints

4.1.4 Support Interface Requirements

4.2 Functional Requirements

4.2.1 Functional Method Constraints

4.2.2 Functional Design Requirements

4.3 Support Requirements

4.3.1 Support Method Constraints

4.3.2 Support Requirements

5 VERIFICATION OF REQUIREMENTS

We have to know that we designed what we were required to design by this document. This process is called 'verification' and it answers the fundamental question: "Did we build what we said we would build?". We verify requirements by testing to see if the requirements are met. Design teams must test every requirement in order to prove that the requirement is met. Testing each requirement means that each requirement must cast in some quantifiable way. In this section you specify how you will test each requirement to show that it has been met.

Don't worry! It is not as bad as it sounds. Sometimes (not always!) you can verify or test a requirement simply by looking at the completed system to make sure some required thing is present.

Testing often 'takes it on the chin' in terms of project schedule. Since integrated system testing typically occurs near the end of a project, the time for testing is compressed against the deadline. People start short-cutting tests to stay on schedule. Sometimes you may get away with it but it is never a good idea either technically or ethically. Epic failures have occurred because of truncated testing. One such failure occurred during the testing of the Hubble Space Telescope. The following is an excerpt from the the official report detailing the failure.

> *Reliance on a single test method was a process which was clearly vulnerable to simple error. Such errors had been seen in other telescope programs, yet no independent tests were planned, although some simple tests to protect against major error were considered and rejected. During the critical time period, there was great concern about cost and schedule, which further inhibited consideration of independent tests.*
>
> *The Hubble Space Telescope Optical Systems Failure Report-NASA November 1990*

If you are interested the whole report is available at (`https://www.ssl.berkeley.edu/~mlampton/AllenReportHST.pdf`).

The Hubble error wasn't caught until the telescope was deployed in space. Can you imagine the cost of fixing this problem? It is not simply a case of bundling you off with your instruments and putting you up in a fancy hotel for a week or two. Some estimates set the price at about $1 billion.

The Dilbert comic strip has a similar, and darkly amusing, view of testing truncation.

(`http://dilbert.com/strip/2010-08-21`)

(http://dilbert.com/strip/2009-07-01)

The key to completing this section is that every requirement has an associated test. The best practice in this section is to match the sub-paragraph numbers in the previous section to the sub-paragraph numbers in this section, e.g the requirement in 4.3.1.6 is covered by the test described in 5.3.1.6.

Possible verification methods include:

1. *Inspection:*
   Inspection is a method of verification consisting of investigation, without the use of special laboratory appliances or procedures, to determine compliance with requirements. Inspection is generally nondestructive and includes (but is not limited to) visual examination, manipulation, gauging, and measurement.

2. *Demonstration:*
   Demonstration is a method of verification that is limited to readily observable functional operation to determine compliance with requirements. This method shall not require the use of special equipment or sophisticated instrumentation.

3. *Analysis:*
   Analysis is a method of verification, taking the form of the processing of accumulated results and conclusions, intended to provide proof that verification of a requirement has been accomplished. The analytical results may be based on engineering study, compilation or interpretation of existing information, similarity to previously verified requirements, or derived from lower level examinations, tests, demonstrations, or analyses.

4. *Direct Test:*

   Test is a method of verification that employs technical means, including (but not limited to) the evaluation of functional characteristics by use of special equipment or instrumentation, simulation techniques, and the application of established principles and procedures to determine compliance with requirements.

5.1 Verify Coverage of Stakeholder Requirements

The tester verifies that everything that the stakeholders have asked for are covered by one or more requirements. It is a good idea for the requirements author(s) to perform a similar check at this point. The tester is likely to do his own analysis or disagree on points in yours, but the exercise itself is valuable. And if you do the analysis you might as well write it down here.

5.2 Interface

5.2.1 Functional Interface Constraints

5.2.2 Functional Interface Requirements

5.2.3 Support Interface Constraints

5.2.4 Support Interface Requirements

5.3 Functional Requirements

5.3.1 Functional Method Constraints

5.3.2 Functional Design Requirements

5.4 Support Requirements

5.4.1 Support Method Constraints

5.4.2 Support Requirements

A tabulation of all the requirements and the testing method with a blank space for results is useful for whomever is doing the testing.

Paragraph Number	Test Type	Tester's Name	Pass/Fail	Date

 *Now read over your completed specification and make additions and corrections. Find others who will be willing to read and comment on the specification (hopefully they will still like you when they are done). The more eyes the better. Ask yourself if you handed this spec to a competent classmate what would they build?*

Congratulations! You have written an engineering specification and that is no mean feat.