Software Specifications

David Duncan

March 23, 2012

Executive Summary

- Creating a software specification is traditionally done via the Software Requirements Specification (SRS) but there has been a growing movement towards the use of “user stories”, which with Behavior-Driven Development are even part of the source code for the customer’s acceptance testing
- Whatever the mechanism, software specification is important, but difficult
- This presentation is a brief introduction to the topic, providing a high-level comparison of the traditional with the cutting-edge, with some tips thrown in as one gets started

Introduction

- Requirements are traditionally the medium for a customer to convey their product needs to the implementing engineers
- Requirements are critical: it is the area where the interests of all stakeholders overlap the most
- A Software Requirements Specification (SRS) document is the common vehicle for capturing requirements
  - This document or one like it is a key part of the classic Waterfall software lifecycle.
- Agile and Behavior-Driven Development (BDD, also called "Specification by Example") have advanced the idea of reducing formal documentation and even focusing on specifications that double in usefulness as acceptance testing.

Requirements are Difficult!

- Software requirements is a sub-field of software engineering that can be a full-time position (i.e., as a "Requirements Analyst").
- Writing requirements can be hard
  - Wording needs to unambiguously express full intent of requirement
- But discovering requirements can be even harder
  - Customers seldom have a great idea of what they want at the level of detail necessary to develop what they ultimately will want
- "The hardest single part of building a software system is deciding precisely what to build." – Fred Brooks, in his essay "No Silver Bullet"

…But Requirements are Important!

- If you've worked closely with software, you probably already have experience with significant time taken to re-write or even re-design something due to a change in direction
  - "If only I had understood that at the start!"
- Studies on this have shown that a defect found in the testing phase of a Waterfall process can be ~10 times more expensive to fix than if found at the requirements phase. [Grady 1999]
- Of course, not all bugs are found in the testing phase—especially when the problem originated with the requirements. The same study indicated a defect found after the delivery (again, in a Waterfall process) ~100 times more expensive to fix than if identified in the requirements phase.
- "No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later." – Fred Brooks, "No Silver Bullet"

Customer Involvement

- Since requirements are so important and since they are ultimately just a definition of what the customer wants, the best way to ensure good requirements is the heavy involvement of the customer.
- To emphasize: It is **critical** to involve the customer with the development of the requirements.
- This is one of the primary tenants of any agile methodology.
- This is also a reappearing theme presented by Karl Wiegers [1] in the context of a more “traditional” SRS before Agile became mainstream, as we’ll see in his list of customer responsibilities presented shortly.

What Does Customer Involvement Look Like? (1 of 4)

- Customer involvement can take different forms
  - The customer may provide their own requirements and (with the help of the customers) the developers may create an SRS from these originating requirements
    - This helps allows the developers to integrate the originating requirements into their company's process and helps to make sure they understand the originating requirements
    - Customer review of the SRS is crucial to make sure the derived requirements are done so correctly!

What Does Customer Involvement Look Like? (2 of 4)

- Customer involvement can take different forms (cont.)
  - The customer and developers may work together on an SRS
    - Most likely, the developers will write the SRS and the customer will (should) at a minimum answer questions and provide multiple iterations of review
  - With agile methods, there is no SRS. Instead, in an attempt to increase emphasis on customer wants and decrease documentation, there is a master story list
    - Instead of requirements, there are user stories, each of which describes a desired functionality

What Does Customer Involvement Look Like? (3 of 4)

- Customer involvement can take different forms (cont.)
  - Behavior-Driven Development (BDD) builds upon Agile ideas
    - Its based upon the Test-Driven Design (TDD) approach of Agile
    - But instead of the previous user stories, use cases are written as natural-language statements that, with the help of some behind-the-scenes definitions, can actually be executed for acceptance testing (generally the last phase of testing, in which the customer validates the provided product)
      - The idea is to encourage the customer to participate in creating the actual acceptance tests

What Does Customer Involvement Look Like? (4 of 4)

- In the examples given, the common thread is that the customer provides--at a minimum--substantial input and review so as to set the project on a proper course
- …But the customer cannot completely disappear once an initial requirements phase is completed

Requirements Will Change (1 of 2)

- It is inevitable that requirements will at some point change
- Hence, customer involvement does not stop after an initial requirements phase, because—again at a minimum—the customer must update the requirements or there is basically no chance the product will meet the customer needs
- Agile methodologies are largely designed upon this very fact of inevitable change
  - They attempt to force the identification of changes by iterating the software often and putting it in the customer's hands

Requirements Will Change (2 of 2)

- With the Waterfall process, change can be a bit more disconcerting
  - Part of this is to a certain degree simply the mindset
    - "Well, we didn't budget for *that*…."
    - "Well, we already built up this nice elegant architecture… that excludes the possibility of that." (Agile methods would encourage the most direct path to next feature.)
  - Nonetheless, in a Waterfall-like process it is important to establish traceability
    - At the most basic level, this means linking integration and acceptance tests to the requirement(s) that spawned them so that when a requirement changes, proper validation can still be done

Wieger's Responsibilities As The Customer (1 of 2)

- 1: "To Educate Analysts and Developers About Your Business"
- 2: "To Spend the Time to Provide and Clarify Requirements"
- 3: "To Be Specific and Precise About Requirements"
- 4: "To Make Timely Decisions"
- 5: "To Respect a Developer's Assessment of Cost and Feasibility"
- 6: "To Set Requirement Priorities"
- 7: "To Review Requirements Documents and Evaluate Prototypes"
- 8: "To Promptly Communicate Changes to the Requirements"
- 9: "To Follow the Development Organization's Change Process"
- 10: "To Respect the Requirements Engineering Process the Analysts Use"

Wieger's Responsibilities As The Customer (2 of 2)

- While Wiegers was published on the topic and known well before the advent of agile methodologies, note the similarities in message to those of Agile
  - He even emphasizes setting requirement priorities; at some places it's just expected everything will be validated correctly!

What Do Specifications Look Like? (1 of 3)

- In Agile and BDD, there is less emphasis on the formality of the specification
  - With a Waterfall-like process, SRSes are commonly a "deliverable" along with other formal documentation and the software product itself, so a formatted, feature-rich document would be typical
  - With agile processes, the standard deliverable is just the software product itself
- With BDD, specifications exist as user stories most often in plain text so that software can use these user stories to run as acceptance tests

What Do Specifications Look Like? (2 of 3)

- With BDD, specifications largely consist just of the user stories themselves (often spread across text/source files).
- When using an SRS, there are typically many of the below sections:
  - Introduction
    - Purpose
    - Scope of the document
    - Definitions
    - System overview
    - References
  - Overall Description
    - Product perspective
    - Product (high-level) functions
    - Operating environment
    - Constraints, assumptions, dependencies

What Do Specifications Look Like? (3 of 3)

- When using an SRS, there are typically many of the below sections (cont.):
  - Requirements
    - External interface requirements
      - These are often mostly captured in a separate document such as an Interface Requirements Specification (IRS)
    - Functional requirements
    - Performance requirements
    - Design constraints
    - Other requirements: safety, security, etc.

What Do Requirements Look Like? (1 of 2)

- In an SRS:
  - Functional requirements
    - E.g., “The system shall log all output voltages.”
  - Performance requirements
    - E.g., “The system shall log voltages at a rate of at least 500 Hz.”
  - Design constraints
    - E.g., “The system shall not incorporate Microsoft Windows.” [Per, for example, a division directive]

What Do Requirements Look Like? (2 of 2)

- As a user story in a Behavior-Driven Development environment:
  - In a text file:
    - Scenario: Start system
      - » Given the system state is Ready
      - » And the user's terminal is the Master terminal
      - » When the user presses the Start button
      - » Then the system should begin dumping program output to the user's screen

Characteristics of Good Requirements (1 of 2)

- Both requirements and user stories should be:
  - Clear, accurate, and feasible to implement
  - Sufficiently detailed but not excessive
    - It is common to encroach a bit into "design territory" while filling out the specification
      - Requirements are the "what", not the "how"
        - » Don't include unnecessary details that should be in a design document or in the design
        - » You might change your mind on the "how" later for reasons totally unpredictable!

Characteristics of Good Requirements (2 of 2)

- Both requirements and user stories should be (cont.):
  - Verifiable
    - Requirements that can't be reasonably tested are very easily:
      - Time-sinks
      - Sources of confusion
      - A recipe for delivering a product different from what the customer really wanted
    - Qualitative adjectives and even absolutes are generally giveaways to reconsider the wording

Verifiable Requirements—An Example

- Consider the following requirement:
  - "The system shall be free of critical failures."
    - Such requirements can and sometimes do flow down from the customer
    - It sounds like a good idea—but how do you test this?
      - The contractor may decide to run a 24-hour unattended test…but later get feedback from the customer that two weeks of operation under numerous states is required for adequate verification.
    - Better: "The system shall experience no lapses in logged data greater than one second under steady-state operation for three continuous days following system start."
      - Now there is little room for ambiguity, at least.

Risks related to Specification Development (1 of 2)

- Insufficient Specification
  - Perhaps management thinks there isn't time or the product "isn't that complex"
- Requirements Creep
  - Requirements/features will change and requirements/features will get added that weren't thought of initially
  - This almost unavoidably adds to the workload
  - Common responses:
    - In a Waterfall process, more time and money is requested over the initial contract. If this doesn't happen, something (often quality) suffers.
    - In an Agile process, it's a bit better defined: scope is dropped on the basis of priority. More time and money can be used to provide a fuller product.

Risks related to Specification Development (2 of 2)

- Ambiguity
  - Agile methods attempt to minimize this impact, but it still exists if the customer intentions are not understood sufficiently early on
- Gold Plating
  - It's not uncommon for programmers to implement functionality not in the requirements
    - Often with good intentions, this usually still uses time and money
- (And, of course…) Insufficient Customer Involvement
  - As mentioned previously, it's very unlikely to end up with the intended product if there isn't sufficient communication of what that product should be

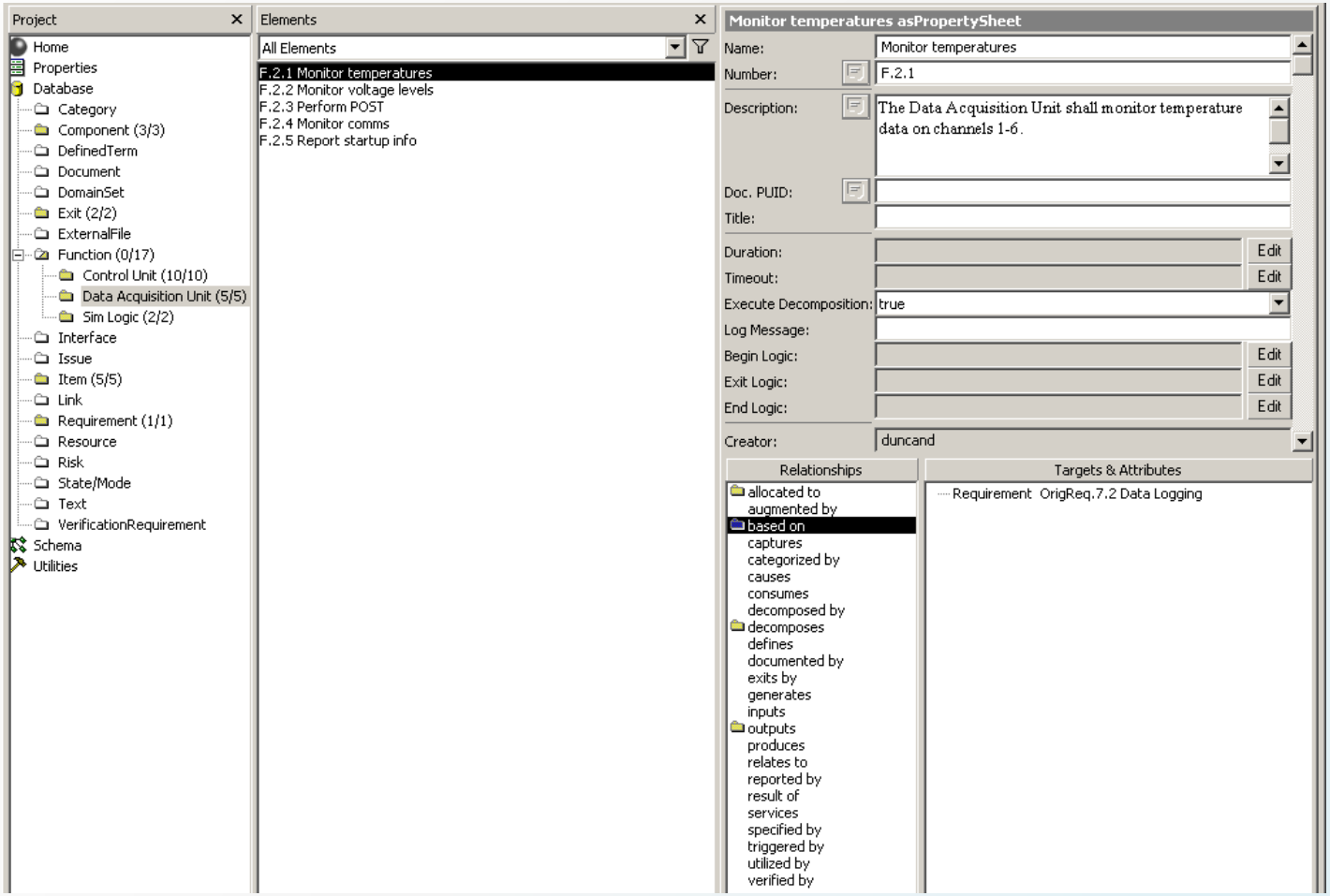
Generating an SRS (1 of 5)

- As mentioned, SRSes often have more sections and formatting than user stories
- SRSes can be created in a word processor
- However, it is not uncommon on larger and/or government projects to use a requirements management tool
  - E.g., IBM's DOORS or Vitech Corporation's CORE

Generating an SRS (2 of 5)

- The following slide shows part of a small example project in CORE
- Different specification elements are collected in a database-like environment
  - Each element has attributes
    - E.g., a name, number, description, change history
  - Each element has relationships
    - E.g., a function is “allocated to” a component, or a function “decomposes” a parent function
- Once the data is properly input into the tool, a script is run on the project to generate an rich-text format document that is the SRS
  - To the extent that other documents such as interface specifications or test plans use the same information, that information only needs to be input once

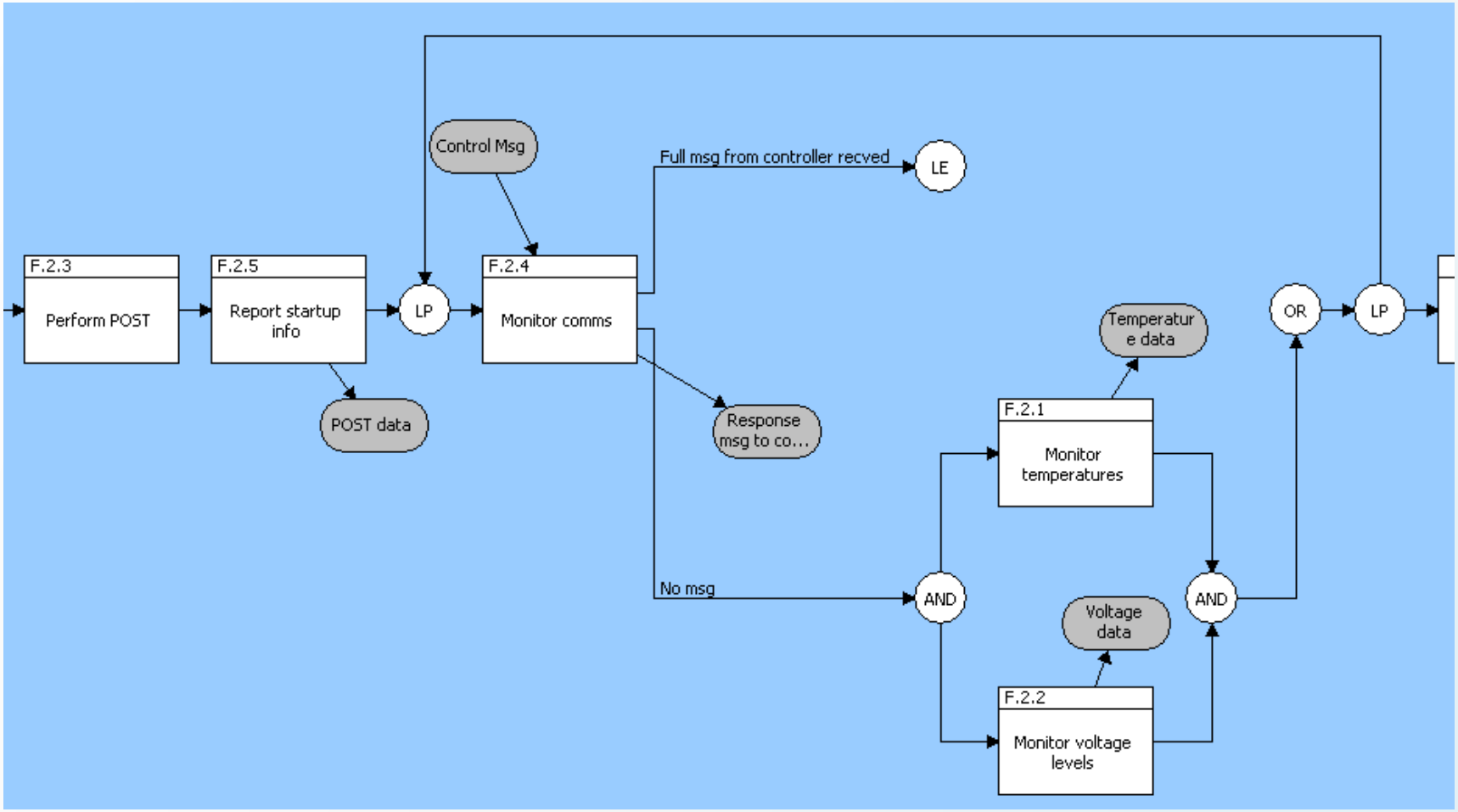
Generating an SRS (3 of 5)



Generating an SRS (4 of 5)

- CORE can also be used to do functional simulations using these elements
- The following slide shows part of a fairly standard enhanced functional flow block diagram (EFFBD) in CORE
  - With some additional configuration and scripting, the same functions used to generate documentation can be used as blocks in a simulation
    - This can be used to further the understanding of functions and their relationship to one another

Generating an SRS (5 of 5)



Conclusion

- Don’t short-change software specification!
  - The quality of the software specification effort has a large effect upon the success of software projects
  - There’s a lot to it, with a lot of methodologies and tools available

For More Information… (1 of 2)

- For more depth on software requirements:

[1] Wiegers, Karl E. *Software Requirements*. Second edition, Microsoft Press, 2003.

- For more details on a typical SRS:

IEEE 830, "IEEE Recommended Practice for Software Requirements Specifications"

- For more on agile process:

Kenneth Anderson's software engineering lectures: http://www.cs.colorado.edu/~kena/classes/5828/s12/lectures/

- For more about software specification development using Behavior-Driven Development:

Adzic, Gojko. *Specification by Example*. Manning, 2003.

For More Information… (2 of 2)

- To download Cucumber, a tool to implement executable specifications:

http://www.cukes.info/

- For more information about CORE, a (pricey) requirements management program similar to Doors:

http://www.vitech.com/

- (Reference for impact of defects:)

Grady, Robert B. 1999. "An Economic Release Decision Model: Insights into Software Project Management," Proceedings of the Applications of Software Measurement Conference, pgs. 227-239. Orange Park, FL: Software Quality Engineering.