Lecture 18: Requirements Specifications

Kenneth M. Anderson

Software Methods and Tools

CSCI 3308 - Fall Semester, 2004

Requirements Specifications

- Why do we need them?
  - A specification is a clear statement of intent
    - Clear intentions are more easily translated into “sharp” milestones (Brooks, pg. 154-155) that are easy to track and evaluate
    - A specification should be as specific and detailed as possible
  - A specification is a contract between a customer and a supplier
    - desktop software: suffers from not having a clear contract specified before the software is developed

Specifications: The Good, the Bad, and the Ugly

- Specifications cover many topics
  - The Good
    - specified conditions of correct operation
      - When the user closes a document window, the associated data file is saved and closed
  - The Bad
    - specified error conditions
      - If the file system reports an error during a save operation, the file’s associated document window is not closed and the user is notified of the error
  - The Ugly
    - unspecified Conditions (!!)

Specifying Input

- Users are unpredictable!
  - Do not specify input that a program may receive
  - Instead, specify a function from input to output
    - F(input) = output
- Example
  - Upon input of an integer from 1 to 100 inclusive, the program will determine if its input is prime and report its results to the user
    - Any input other than an integer from 1 to 100 inclusive, including integers outside the range of 1 to 100, non-whole numbers, and non-numeric input will result in an error message

Discussion

- The example is very specific
  - It defines its legal inputs carefully and specifies illegal inputs explicitly
  - It defines “what” the program should do, not “how” the program should do it
    - Given these requirements you can create several alternative designs that satisfy them
    - For instance, a system that uses speech input and output is perfectly acceptable, as is a system with a graphical user interface or a command line interface

Specifying “What” not “How”

- A requirements specification specifies the **behavior** of an application, not its **implementation**
- Specifying Implementation
  - The program must have a linked list to hold pending alarms. Each alarm in the list is a structure containing the date, hour, and minute the alarm should sound. The list should be sorted according to time.
- Specifying requirements
  - The program shall provide an alarm clock feature. A user can specify multiple alarms. Each alarm rings the computer’s bell when it is activated.

More on specifying behavior

- A requirements specification is the first document created for a program
- Specifying a program’s behavior allows for maximum flexibility during design and implementation
  - It answers the question: “Why am I writing this program?”
  - Specifying implementation first, on the other hand discourages the consideration of alternatives
    - It constrains the design inappropriately

Formal and Informal Specs.

- Specifications can be informal
  - natural language based
    - no matter how hard you try, natural language specifications will always have some degree of ambiguity
- or formal
  - based on a mathematical model
  - typically requires training to use and apply correctly

Example Informal Specification

- The meaning of integer division, div(a, b), is the same as floating point division with the fractional part rounded off towards zero.
- The meaning of modulo, mod(a, b), is the value of the fractional part that would be rounded off by div(a, b).

Example Formal Specification

- div(a, b) = integer q, such that
  - $0 \leq (a - b * q) < |b|$ if $a > 0$
  - $0 \geq (a - b * q) > -|b|$ if $a < 0$
  - 0 if a = 0
- mod(a, b) = a - (div(a,b) * b)

More on Formal Specifications

- Being based on a mathematical model means
  - every symbol is well defined
    - syntax
      - how is a symbol combined with other symbols
    - semantics
      - what is the symbol's meaning, how does it behave
- Formal specs often reuse information
  - In our previous slide, we did not define the greater than, less than, and equals symbols;

Trade-offs between formal and informal specs

- Formal specs are **not** always better than informal ones
  - In early stages of a development project, you may not know or understand enough to create a formal spec
    - an informal spec can serve as a starting point
  - Formal specs are often difficult to understand
    - This can discourage people from using them
    - An informal spec can be used to annotate and explain a formal spec
  - Formal specs are typically expensive to create
    - They require specially trained workers
    - Not all parts of a project need to be formally specified: use formal specs where its absolutely critical that a program behaves as specified; such as flight control software

Brooks' Corner: The Whole and the Parts

- How does one build a successful program?
  - Focus on the specifications and test them!
    - Testing should be preformed by an external group
  - Top-down Design
    - Design as a set of refinement steps
    - Use of abstraction at each level
    - Modular decomposition

The Whole and the Parts, cont.

- Other techniques
  - Structured Programming
  - Component Debugging
  - System Debugging
    - Use debugged components (reuse)
    - Build scaffolding (stubs, test data)
    - Control Changes
    - Add one component at a time, and quantize updates