Lecture 5: Introduction to Specifications

Kenneth M. Anderson

Foundations of Software Engineering

CSCI 5828 - Spring Semester, 2000

Today's Lecture

- Introduction to Specifications
  - Present an extended example
    - *make*

Specification

- Dictionary Definitions
  - specific - that pertaining to a particular species
  - specify - to be specific
  - specification - act of specifying
- Physical Sciences
  - "specific gravity", "specific heat"
    - convey particular properties and characterize the behavior of physical substances in any context of their usage

Specification, cont.

- Engineering and Architecture
  - Specification means "a statement of particulars describing the structural and behavioral details of a product to be developed.
- Software Specification (a narrow view)
  - Denotes "a precise description of a system's objects, a set of methods to manipulate them, and a statement on their behavior for the duration of their existence"

Why do we need specifications?

- Complexity!
  - Software
  - Size
  - Structural
  - Environmental
  - Application Domain
  - Communication

How do specifications help?

- A proper specification can control and adequately contain certain types of complexity
- Without specification, software complexity is uncontrollable!
  - As we shall see when we read Fred Brooks later this semester

Two Conceptual Tools for Specifications

- Abstraction
  - the specification contains only key features, without a description on how they can be realized
- Decomposition
  - ensures that the properties of a system follow from the properties of its parts

Essential Specification Properties

- Define observable system behavior
- Define precise and simple interfaces
- composable (behavior of whole from parts)
- test for conformance
- Analyzable; given a property and a specification, we should be able to prove that the property holds for the system

Essential Specification Properties

- It must be possible to develop a program from the detail design specification
- The design specification must contain a description of all behaviors expressed by the behavioral specification
- It must be possible to test for conformance
- It must be possible to subject a specification to rigorous analysis

Specification Qualities

- Clear, Unambiguous, and Understandable
  - Are these self-contradictory?
    - For instance, being unambiguous often requires a lot of qualifications, which can reduce clarity
- Consistent
  - How can we check this?
- Internally and Externally Complete
  - Does completeness reduce understandability?
  - What about normal vs. exceptional behavior?

Specifications are Software

- Have a *Life cycle*
  - rationale, iteratively refined, used, & enhanced
- Should be *Modular*
  - modularity promotes reuse and high cohesion
- Come in *Versions*
  - you won't get it right the first time…or its requirements will change!
- Exhibit *Dependencies*
  - but we want loose coupling...

Cohesion and Coupling

- These concepts are typically applied to software modules, OO classes, etc.
  - but they actually can be applied to any set of items that exhibit dependencies on each other
- Cohesion
  - how focused an entity is on a particular task
    - e.g. a software module that handles only one task
- Coupling
  - the degree to which objects depend on each other

Cohesion and Coupling, cont.

- We strive for high cohesion...
  - A highly cohesive entity is focused on one task. If the task changes, it impacts its associated entity only
  - Low cohesion means that an entity is responsible for more than one task, or a task is split between entities
    - a change in a task then requires changes in multiple entities or modifications to an entity that is only peripherally related to the changed task

Cohesion and Coupling, cont.

- …and we strive for loose coupling
  - That is, we want a system with low interdependencies
  - In highly coupled systems, a single change may impact multiple entities
  - Therefore a loosely coupled system is more resistant to change, since it propagates to fewer entities

Relationship to Specifications

- Highly cohesive specifications
  - Focused on one aspect of a system
  - If we have a question about that aspect, we go to the one specification for that aspect
  - Systems with similar needs can share the spec.
- Loosely coupled specifications
  - If a specification changes, the impacts of the change are mitigated and/or minimized

Specifications Can Be Wrong

- Need to Validate and Verify (V&V)
- V&V is a "with respect to" Activity
  - Implies existence of another specification
  - But how do we V&V that other specification?
- Human Holds the Ultimate Specification
  - This means that requirements are incomplete, ambiguous, and may change frequently!

Getting Specifications Right

- (Reusable) Layers and/or Modules Help
  - A confidence game
    - If we have used a specification before and it contributed to a successful system, we have more confidence in it than a newly developed specification
- Formality Helps
  - Mathematical models increase our ability to check our assertions for correctness

Specification Notations

- Key to Qualities
- Affect V&V Options
- Most are Equivalent in *Expressive Power*
- Differ in *Expressive Convenience*

Specification/Modeling Styles

- Operational (or Imperative)
  - Described according to desired *actions*
  - Usually given in terms of an *execution model*
- Descriptive (or Declarative)
  - Described according to desired *properties*
  - Usually given in terms of *axioms* or *algebras*
- Structural (or Relational)
  - Described according to desired *relationships*
  - Usually given in terms of *multi/hyper graphs*

An Informal Specification

- A system consists of a set of object files. Each object file is derived from one or more source files. Object and source files have a timestamp indicating when they were last modified. If an object file is older than any source file, then the object file must be rederived.

First Steps

- A system consists of a set of object files. Each object file is derived from one or more source files. Object and source files have a timestamp indicating when they were last modified. If an object file is older than any source file, then the object file must be rederived.

Formalize

- $O = \{o_1, o_2, o_3, \ldots\}$
- $S = \{s_1, s_2, s_3, \ldots\}$
- $F = O \cup S$
- $T: F \rightarrow \Re$
- $D: O \rightarrow \text{PowerSet}(S)$
- $\text{ForAll}(o \in O), \text{ForAll}(s \in D(o)) \; T(o) > T(s)$

- O = set of object files
- S = set of source files
- F = all files
- T = timestamp relation
- D = derived relation
- An assertion: o's timestamp must be greater than the timestamps of D(o)

Make Specification Language

- Hybrid Declarative/Imperative/Relational
- Dependencies are Relational
- Rules are Declarative
- Actions are Imperative