Software Development Phases, their Artifacts, and Processes:
Part 2: The Requirements Phase

Software Engineering
Computer Science 520/620
Spring 2011
Prof. Leon Osterweil

Today’s Problem

How to Build Something Like this

Using some kind of approach like one of the following

Process model

1970’s

• Recognition of feedback loops
  – Confined to successive stages
• “Build it twice”

Early prototyping
To the satisfaction of all of these?

**Our Approach**

- What is the goal/role of each component type?
- What is the nature of it?
  - Eg. what internal structure does it have?
- What sorts of stakeholders are interested in it?
- What sorts of questions do they generally have about it?
  - Internal relations
  - External relations
- What sorts of processes deal with it?

---

**Some Papers in Requirements Engineering**


---

**Requirements: Goals and Purposes**

- Clarify needs before plunging into design
  - Customer “knows” what is wanted
  - But usually doesn’t know how to say it
- Weak sense of what can be achieved
- Clarify acceptance criteria
  - How to know it really delivers what was wanted
  - Serve as guide to developers, testers, customers, maintainers
  - “Baselining” requirements
Requirements Spec. 

Characteristics of system to be built must match required characteristics 

Test Results must match required behavior 

Test Plan 

Design 

Requirements Spec. 

Characteristics of system to be built must match required characteristics 

Test Results must match required behavior 

Test Plan 

Design 

Functional 

Safety 

Robustness 

Performance 

Accuracy 

Test Plan 

Requirements Spec. 

Characteristics of system to be built must match required characteristics 

Test Results must match required behavior 

Test Plan 

Requirements Spec. 

Characteristics of system to be built must match required characteristics 

Test Results must match required behavior 

Test Plan 

Requirements Specification Driven By Stakeholders and their Questions 

- Customers 
  - What must it do? 
- Developers (eg. designers) 
  - What do I have to get it to do? 
- Testers 
  - What is it supposed to be doing? 
  - How would I know it if I saw it? 
- Users 
  - What is it supposed to do? 
- Regulators 
  - Is enough being required? 
- Others???
### Requirements Specification Parts
Help stakeholders organize their thoughts about needs by decomposing requirements specification into categories needs and desires.

Some examples:
- Introduction/Background
- Functional
- Environmental
- Performance
- Accuracy
- Robustness
- Security
- Safety

### Background/Introduction
Purpose: Give background/context out of which the problem arises, and directions in which it is likely to go

- Should contain glossary, references
- Should give intuition about problem, domain, existing solutions, components
- Probably best written mostly in natural language

Example: UMass has 20,000 students, slow growth next few years
- Semester system
- Existing system that works, but is not great
- Define: FTE, fulltime load, etc.

### Functional
Purpose: Indicate the functional transformations that the system will have to compute

- Likely to be large and complex, therefore aids to easier and clearer comprehension are needed (eg. hierarchy)
- Important to state WHAT the functions are and not HOW they are to be computed
- Promising formalisms: dataflow diagrams, FSM’s
- Usually the chief focus of a requirements specification, and of requirements formalisms—but non-functional requirements are often at least as important

### Environmental
Purpose: Indicate the environment in which the software will have to operate

- On which hardware and software will the software run?
- With which other applications will it have to interface?
- What will be the nature of the user community it will have to support?
- With what other manual and automated systems will it have to interface correctly?

Example: System is to be interactive
- Most users to be students
- Must run using cellphones, PDAs
- Must print reports on existing forms
- Must interface successfully with existing student and administrative databases

### Performance
Purpose: Specify how much computer and human resource can be allocated to support the execution of the software

- How much computer memory can the software use?
- How fast must response time be?
  - average case
  - worst case
- How long will users wait for batch runs to terminate?

Example: 2 second response time
- overnight printing of all reports
- 1 Gbytes of memory available
- 500 GBytes of disk available

### Accuracy
Purpose: Specify how much tolerance (if any) is acceptable in the results

- Most important in numerical computations, but...
- Often where "optimality" is defined eg: what is a "good" game of chess?

Example: Reject scheduling constraints that cause more than 10% of all student requests to be denied
### Robustness

**Purpose:** Specify what sorts of abuse the software will have to resist, and how it will respond

- What kinds of "illegal" inputs might be expected, and what should be done about them?
- Must the system fail safe, fail soft? When?
- What abnormal environmental conditions might be expected?

**Example:**
- System must never corrupt any database
  - even after a crash
- System must deny illegal requests politely
- System must not crash due to
  - lack of storage
  - user overload

### Security

**Purpose:** Specify which users can do which things, and when

- Usually there are classes of users—what are they?
- How to distinguish among the users?
- Matrix (?) to specify what accesses and permissions different classes and users will have?

**Example:**
- Students cannot:
  - change course assignments
  - access data on others
- Faculty cannot:
  - cancel courses
  - change course assignments
- Faculty can:
  - access any student data
- Administrators can: .... do pretty much anything...

### Safety

**Purpose:** Specify what hazards must be avoided

- Specify what the software must NEVER be allowed to do
- Has some elements of an inverse or negated set of requirements

**Example:**
- System must never corrupt student information database, or faculty personnel database

### Not All Go Into All Requirements Specs.

- Some of these may be omitted; some emphasized/deemphasized
- Other sorts of requirements may be added/substituted
  - eg: reliability, flexibility, portability....
- Requirements specification provides information needed to satisfy needs of all stakeholders
- Different stakeholder mixes determine choices of what goes into the requirements spec.

### Requirement Specification Challenges

- **Is it Complete?** (to the extent required)
  - Ultimately impossible to be sure about this
- **Is it Consistent?** (no internal contradictions)
  - Many possible interpretations of this
- **Is it unambiguous?** (possible multiple interpretations)
- **Is it sufficiently precise?**
  - It is possible to be too precise too
- **Is it Feasible?**
  - If it asks the impossible it would be good to know it
- **Is it Even?** (consistent levels of detail)
- **Is it Understandable?** (what does that mean?)
  - by all stakeholder groups!
- **Is there an implementation bias?**
- **Is there a good basis for proceeding to design?**

### A Requirement Specification Is Never Perfect in All (Any?) Aspects

- Imperfections are often understandable, tolerable, unavoidable
- Look at real underlying stakeholder needs for the requirements specification (communication, clarity, precision, modifiability....??)
- Plan requirements content, structure, relations to meet these needs
- Requirements specification medium is crucial in helping assure needs are met
- Select requirements specification medium to address needs
Requirements Specification
Formalisms and Processes

- Support addressing these challenges
- Different formalisms emphasize facilities for supporting the answering of different types of questions
- Different processes provide to different degrees of assurance and thoroughness

Requirement Processes

- Elicitation
  - How to ascertain requirements
  - Interviewing, classifying, organizing
  - Emphasis on perspectives/viewpoints
- Review
  - How to determine consistency, completeness, etc.
  - Emphasis on analysis
  - Need for semantic basis and inference reasoning
- Revision/Improvement/enhancement
  - How to add, delete, modify
  - Rereview: how to determine consistency of modified requirement specification

Being Precise about Requirements Processes

- Helps developers understand
  - Other stakeholders too
- Basis for automated support
- Being precise about processes is closely tied to being precise about artifacts

Example Requirement Specification Process

Definition of Define Rqmt Element

Better Definition of Define Rqmt Element
Many Details Left Out
Many Other Alternatives

- When to check what
- How to do the checks
- How to respond
  - And when
- Etc.
- Being more specific about process entails being more specific about artifacts/products

Focus on artifact specification approaches and issues

Evaluation of Requirements
Specification Media

Representative Evaluation Criteria:
- UNAMBIGUOUSNESS
- CLARITY
- COMPLETENESS
- VERIFIABILITY
- CONSISTENCY
- MODIFIABILITY
- LIFECYCLE UTILITY

How Do They Match Up with Each other?
Natural Language Prose
Requirements Specification

- Write requirements in "plain English"
- Build upon universal base of understanding of natural language
- Possible to augment with defined terms
- Use of punctuation for clarification
- Text and word processing systems help automate/maintain/alter

Examples:
- All input data sets will be terminated with an end of file record
- System will respond to service requests within 2 seconds
- System will have a friendly user interface
- System will never go into an infinite loop

Problem: How to reason about a natural language reqts. spec?
How to determine: completeness, unambiguity, etc.?

Disciplined Use of Natural Language

- Natural response to problems of:
  --imprecision
  --ambiguity
  --consistency (especially when due to size)
- Familiar approaches:
  --Restricted use of defined terms
  --Introduction of structuring (paragraph numbering, outline form, templates, etc.)
- Other, earlier examples of disciplined use of natural language:
  --Legal documents
  --Recipes

Data Base Approaches

- Requirement items stored as database entries
- Queries to retrieve information
- Database tools to check for consistency

Examples:
- Use of structure and reserve words
- User interaction functions:
  **Timing:** All functions must execute in < 2 seconds
  **Subfunctions:**
  - Query
  - Browse
  - Enter
- Disciplined use of naming
  ....input_value: pay_rate
  pay_rate: input_to ......

PSL (Relational Database Organization)

**DESCRIPTION:**
this process performs those actions needed to interpret
time cards to produce a pay statement for each hourly employee.;
**KEYWORDS:** independent;
**ATTRIBUTES ARE:**
  complexity-level high;
  GENERATES: pay-statement, error-listing;
  RECEIVES: time-card;
  SUBPARTS ARE: hourly-paycheck-validation, hourly-emp-update,
  h-report-entry-generates, hourly-paycheck-production;
  PART OF: payroll-processing;
  DERIVES: pay-statement;
  USING: time-card, hourly-employee-record;
  DERIVES: hourly-employee-report;
  USING: time-card, hourly-employee-record;
  DERIVES: error-listing;
  USING: time-card, hourly-employee-record;
  PROCEDURE: <<not usually included in a requirements spec.>>
  HAPPENS: number-of-payments TIMES-PER pay-period;
  TRIGGERED BY: hourly-emp-processing-event;
  TERMINATION-CAUSES: new-employee-processing-event;
  SECURITY IS: company-only;

Hierarchical Decomposition Organization

- Requirements Specification as hypertext
- Structure (DAG) of Requirements Elements
- Child element represents part-of relation
- Requirement Element is a record
- Requirement Element fields carry information as:
  - Instances of preset types
  - Instances related to others by relations
    -- express consistency rules
    -- define consistency determination
    -- define inconsistency remediation
  - Relations among
    -- Requirement elements
    -- Requirement elements and parts of other artifacts
      (e.g., testplan elements, other reqts. representations)
Partial Formalization of Functional Decomposition Requirement Specification

A Requirements Specification, R, is a set
\[ R = \{ r \mid r \text{ is a requirement element} \} \]
r, a requirement element, is a tuple,
\[ r = (\text{children}(r), \text{parent}(r), \text{timing}(r), \text{functionality}(r), \text{robustness}(r), \text{inputs}(r), \text{outputs}(r) ) \]
ETC.

Physiology of a Requirement Element

Example: Using this to specify requirements for a Multifunction Watch

- Watch functions include
  - Telling time
  - Alarms
  - Telephone directory
  - Appointment book
  - Memo pad

- With various additional constraints
  - Speed
  - Accuracy
  - Robustness
  - Etc.

Partial Formalization of Functional Decomposition Requirement Specification

A Requirements Specification, R, is a set
\[ R = \{ r \mid r \text{ is a requirement element} \} \]
r, a requirement element, is a tuple,
\[ r = (\text{children}(r), \text{parent}(r), \text{timing}(r), \text{functionality}(r), \text{robustness}(r), \text{inputs}(r), \text{outputs}(r) ) \]
ETC.
Using this for Hierarchy Specification

children(r) = \{ s \in R | s is a subfunction of r \}

Question: What do we mean by “subfunction”?  
--Is included textually within  
--Requires in order to execute properly  
(i.e. is a module that is used)  
-- ??

parent(r) = The s \in R  such that r \in children(s)  
ancestors(r) = the transitive closure of r under the  
parent relation  
= r U parent(r) U parent(parent(r)) U ....

Timing Requirement Specification

timing(r) is a Boolean function

Intuition: To help clarify, define timing(r) to be  
the function pass(testresult), which maps  
the set of results of executing testcases for r  
onto the Boolean values (True, False)

Thus, pass is a function that defines what  
means for a testresult to “pass” (i.e. to  
satisfy a timing constraint)

Examples: timing(datebook)  
= pass_{timeout}(testresult) \iff testresult < 0.1 seconds  
= pass_{time}(testresult) \iff testresult < 0.1 + \frac{\text{size(datebook.cont)}}{1000}

Note obvious proximity to testing concerns

Functionality Specification

functionality(r) \equiv n, a node in a graph, G

Intuition: The functionality being specified here is  
being specified with the help of a graph, whose  
semantics provide rigor to the definition of the  
function being required.

The semantics of the graph will turn out to be  
vitaly important in supporting reasoning about  
the requirements specification and its consistency  
with design specifications.

Using a DFD

functionality(r) \equiv n, a node in the DFD, D = (N, E)  

Intuition: The functionality being specified here is  
being specified with the help of a DFD.

Input/Output Specification

inputs(r) = \{ i | i is an object needed for the execution  
of the function denoted by n, the  
DFD node used to define the  
functionality of r \}

outputs(r) = \{ o | o is an object created by the execution  
of the function denoted by n, the  
DFD node used to define the  
functionality of r \}
An FSM can help too

- Use FSM to define how a (set of) function(s) must be related to others.
- Create an “Accessed By” field?
- Value is a pointer into FSM that describes the workings of the system

Each node (state) pointed to by the appropriate field in the appropriate node of the requirements definition

Children of these nodes point to these “substates”

Multirepresentation Systems

- Have seen that different representations are of different uses
- One diagram may be useful in different ways to different stakeholders
- But most stakeholders require a variety of diagrams
- Several different diagrams can be expected to be needed to satisfy the different stakeholders
- Problems with different views/diagrams
  - Are they all representing the same software product?
  - How to assure that they are all consistent with each other?
  - If the product changes, then ALL views must change correspondingly

STATEMATE supports Requirements Specification

- Key feature is maintenance of consistency among views
  - Done by projecting views of abstract model
  - Change abstract model through changes to views
- Use of Hierarchical Decomposition for understandability
- Different diagrammatic views
  - Statecharts (Enhanced FSMs)
  - Enhanced Data Flow Diagrams
- Statecharts, FSMs both support specification of different requirements types simultaneously
  - Functional
  - Robustness
  - Safety
Statechart

Robustness Specification

robustness(r) = a first order logic implication
A_r => B_r such that whenever an execution
of the functionality defined by r “satisfies” the
predicate A_r, then it necessarily also “satisfies”
the predicate B_r.

Example: A_r: The clock stops ticking
B_r: The datebook keeps working

Security Specification

• Who is allowed to use this function?
• Under what circumstances?
• With what restrictions?
• Etc.

This Diagram Suggested When to Check
Examples of what to check and how

**Internal Consistency:**
\[ \forall r \in R, s \in \text{children}(r) \Rightarrow \text{parent}(s) = r \]
\[ \forall r \in R, \forall \text{testresult}, \text{pass} (\text{testresult}) = \text{True} \]
\[ \Rightarrow \text{pass} (\text{testresult}) = \text{True} \]
\[ \forall s \in \text{descendant}(r) \]

**Interartifact Consistency:**
\[ \forall r \in R, \forall i \in \text{inputs}(r) \Rightarrow i \text{ is an input to the node } n \text{ in the DFD that defines the functionality of } r \]

---

**Part of this feature of Process Definition**

**Checking Timing Requirements Against Design (a look ahead)**

Showing consistency between a timing requirement, and timing characteristics of a design specification:

Suppose design is specified by a flowgraph; flowgraph is defined by means of the Immfol relation. Use Immfol to derive the Fol relation.

Suppose each flowgraph node is annotated with a timing specification. Suppose requirements node is defined by one or more flowgraph nodes.

Trace flowgraph paths to compute timing of function, and compare to requirement node timing spec.

---

**Other Requirements Structures**

- Functional decomposition often works well as the structure of a requirements specification
  - Key feature: “all” requirements are clustered by function
- Other decompositions are sometimes better
- Goal decomposition is increasingly interesting
- vanLamsweerde’s KAOS system
  - Goals decomposed into subgoals
  - Special attention to robustness with scoped exception specification
Security Decomposition

• Focus on who can do what
• Hard to infer that from the functional decomposition
• Makes most sense when security is the key focus of the system
• May make it harder to infer other things
  – Presumably they are less important

(Recall) Typical Stakeholders and their Needs

• Customer: needs to
  – communicate what the system must be like
  – have a basis for determining if it works right
• Developer: needs to
  – understand what the system must be like
  – what the design must implement
• Tester: needs to
  – know how to evaluate the final system
• User: needs to
  – see that real underlying needs are going to be met

Evaluate These Requirement Specification Approach Against Requirements Problems

• Incompleteness
• Inconsistency
• Ambiguity
• Infeasibility
• Unevenness
• There are others....

Requirements Incompleteness

• EXAMPLES: Missing requirements, missing details
• CAUSES
  – Customer may be unavailable, inaccessible, a group
  – Customer asks for too little
    – doesn't understand computer's capabilities
    – doesn't understand interfaces to larger system
  – Customer doesn’t think of everything
    – desired function not mentioned
    – special case forgotten
  – The world changes, new things are required
• REMEDIAL APPROACHES
  – Cross checking requirements with each other
  – Facilitated by building in relations and redundancy

Requirements Inconsistency

• Many types of inconsistency: inconsistent wrt what?
  – Timing: function/subfunction mismatch
  – Functionality: subfunctions don’t “flow right”
  – Robustness: no specification of error recovery
• CAUSES
  – Customer may be a group that disagrees
  – Different people may negotiate different parts
  – Early identification of inconsistency can be a big benefit
• REMEDIAL APPROACHES
  – “Cross-checking” related requirements elements
    – What to check against what? How? For what?

Requirements Ambiguity

• More than one possible set of inferences
• CAUSES
  – Customer may be a group where noone sees the whole picture—at least at first
  – Difficult to spot ambiguity in large, complex applications
  – So many parts, related to each other in so many ways
• REMEDIAL APPROACHES
  – Materialize the relations
  – Use them to identify inconsistencies
• PROBLEM: How to control which inferences are possible, and which are not allowable?
Requirements Infeasibility

- **CAUSES**
  - Customer asks for too much
    - no conceivable algorithm
    - unrealistic requests
  - Still useful to know what ultimate desires are
    - Enables early expectation management
    - Suggests incremental development planning
- **REMEDIAL APPROACHES**
  - This is a very hard problem
  - Need to determine consistency of requirements specification with designs and implementations
  - Multiple notions of “consistency”

Requirements Unevenness

- **CAUSES**
  - Different sources of information
  - Different people write different parts
  - Different parts of specification are more difficult than others
- **REMEDIAL APPROACHES**
  - Relate details at different levels to other details at similar levels