Requirements

Software Engineering
Computer Science 520/620
Spring 2014
Prof. Leon Osterweil
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?
- What sorts of relations must it participate in?
  - Internal relations
  - External relations
- What sorts of processes deal with it?

These phases are not necessarily executed in any particular order
Some Papers in Requirements Engineering


Requirements: Goals and Purposes

Why “do requirements”?

- 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.

Test Results must match required behavior

Characteristics of System to be built must match required characteristics

Design

Test Plan

Requirements Spec.

Functional
Safety
Robustness
Performance
Accuracy

Design

Modules
Components
Constraints
Design Decisions
Components
Characteristics of system to be built must match required characteristics.

Requirements:
- Functional
- Safety
- Robustness
- Performance
- Accuracy

Design:
- Modules
- Components
- Constraints
- Design Decisions
- Components

Testplan:
- Inputs
- Outputs
- Setup
- Timing
- Knockdown
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 of needs and desires.

Some examples:

• Functional

• 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
128 Mbytes available on PDAs
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 data must be protected, in what ways, from whom, etc.

Usually there are classes of users--what are they? How to distinguish among the users?

Categories of data too.

Matrix (?) to specify what accesses and permissions different classes and users will have?

Example: Students cannot:
--change course assignments
--cancel courses
--access data on other students

Faculty cannot:
--cancel courses
--change course assignments

Faculty can: access some student data: which?

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 divulge credit card data

System must never divulge phone contact data

System must never divulge address or data to unauthorized parties
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.

SOME EXAMPLES OF THESE UNDERLYING NEEDS:

- Communication
- Testability
- Precision
- Clarity
- Completeness
- Changeability

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 unambiguos? (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

Evaluation of Requirements Specification Media

Representative Evaluation Criteria:

- UNAMBIGUOUSNESS
- CLARITY
- COMPLETENESS
- VERIFIABILITY
- CONSISTENCY
- MODIFIABILITY
- LIFECYCLE UTILITY
Evaluation of Requirements
Specification Media

Representative Evaluation Criteria:

• UNAMBIGUOUSNESS
• CLARITY
• COMPLETENESS
• VERIFIABILITY
• CONSISTENCY
• MODIFIABILITY
• LIFECYCLE UTILITY

Example Specification Media:

• NATURAL LANGUAGE
• STRUCTURED NATURAL LANGUAGE
• DIAGRAMS/CHARTS (DFD’s, FSA’s, Petri Nets)
• FORMAL APPROACHES
• COMBINATIONS OF THE ABOVE

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

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;
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** ..... 

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 rqts. representations)
Functional Decomposition Rqts. DAG

Requirement Element: An Example Structure
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

\[ R=\{ r_i | r_i \text{ is a requirement element} \} \]

\( r \), a requirement element, is a tuple,

\[ r = ( children(r), parent(r), timing(r), functionality(r), robustness(r), inputs(r), outputs(r) ) \]

ETC.
Physiology of a Requirement Element

<table>
<thead>
<tr>
<th>NAME</th>
<th>ROBUSTNESS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DATE</th>
<th>INPUTS</th>
<th>CHILDREN</th>
<th>OUTPUTS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PARENTS</th>
<th>ACCURACY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LOCAL DATA</th>
<th>TIMING</th>
<th>FUNCTION DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Using this for Hierarchy Specification

\[ \text{children}(r) = \{ s_i \in R \mid s_i \text{ is a subfunction of } r \} \]

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

\[ \text{parent}(r) = \text{The } s \in R \text{ such that } r \in \text{children}(s) \]

\[ \text{ancestors}(r) = \text{the transitive closure of } r \text{ under the parent relation} \]
\[ = r \cup \text{parent}(r) \cup \text{parent(parent}(r)) \cup \ldots. \]

Timing Requirement Specification

What kind of notation should be used to specify timing requirements?
Timing Requirement Specification

timing(r) is a Boolean function

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

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

`timing(r)` is a Boolean function

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

Thus, `pass_r` is a function that defines what it means for a `testresult` to “pass” (ie. to satisfy a timing constraint)

Examples: `timing(datebook) ≡`  
`pass_{datebook}(testresult) ≡ testresult < 0.1` seconds  
`pass_{datebook}(testresult) ≡ testresult < 0.1 + size(datebook_cont)/1000`

Note obvious proximity to testing concerns

Functionality Specification

`functionality(r) ≡ 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 vitally important in supporting reasoning about the requirements specification and its consistency with design specifications.
Using a DFD

functionality(r) ≡ 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) = { in_i | in_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) = { out_j | out_j 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

Diagram:

- Time display mode
- Datebook mode
- Alarm display mode
- Phone book mode

Buttons:
- Press button A
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

- Initialize
  - do: Initialize course object

- Unassigned
  - do: Assign professor to course
  - numStudents = 0

- Canceled
  - do: Send cancellation notices

- Closed
  - do: Report course is full

- Open
  - entry: Register a student
  - registration closed
    - numStudents < 3
  - registration closed
    - numStudents > 3

- addStudent
  - addStudent/numStudents = 0

- RegistrationComplete
  - do: Generate class roster

- cancelCourse
Statechart with Nested States

Robustness Specification

What kind of formal notation might be used to Specify robustness requirements?
Robustness Specification

*Perhaps a first order logic implication?*

\[
\text{robustness}(r) = \text{a first order logic implication}
\]

\[
A_r \Rightarrow B_r \text{ such that whenever an execution of the functionality defined by } r \text{ “satisfies” the predicate } A_r, \text{ then it necessarily also “satisfies” the predicate } B_r.
\]

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.

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
Some Possible Relations Involving Such Nodes

Parents of Children = Children of Parents

Parents of Children

Children of Parents

Speed of Children > Speed of Parents

Speed of Children

Speed of Parents

Functionality related to testcase inputs/outputs

Same function as one in a DFD

Data object also used in a DFD

Local Data

Function Definition

Functionality related to testcase inputs/outputs

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
A Different Approach: Use Case Structuring in Tokeneer

- Tokeneer project to build high-security software
  - To control access to a facility
- Software has little functionality
- But stringent security requirements
- Developed to assure that it works “right”
- Requirements developed using
  - Use Cases
  - Then Z

What is Tokeneer?

- Software system for controlling access to a high-security area
- Developed so that security and other requirements could be formally verified
- Requirements specification is contained on a large document
  - Approach is mainly through Use Cases
Tokeneer Requirements Specification

- Functional (behavioral) requirements through eleven (11) Use Cases
  - Each use case covers a key scenario
- Some supplementary diagrams as well
  - Some for non-functional requirements

5 Behavioural Requirements

The required behaviour of the ID Station is specified here using scenarios, which run through typical uses of the ID Station and define the interaction between the ID Station and its connected systems. In each case the scenario focuses on a successful outcome, but it also covers various conditions that may arise that do not allow the successful outcome to be achieved. The full behaviour of the system, including both successful and failed outcomes, constitute the system requirements.

The scenarios considered are:

1. User gains allowed initial access to Enclave
2. User is denied prohibited initial access to Enclave
3. User gains allowed repeat access to Enclave
4. ID Station is started and enrolled with input from the Enrolment Station
5. ID Station is started already enrolled
6. ID Station is shut down
7. Security Officer updates the configuration of the ID Station
8. Audit log is archived
9. Guard manually unlocks the door
10. Administrator logs on
11. Administrator logs off
5.2 Scenarios

5.2.1 User gains allowed initial access to Enclave

Description

A User who should be allowed access to the enclave is given access, making use of biometric authentication.

Stimulus

User inserts a smartcard into the smartcard reader.

Assumptions

ScGainInitial.Ass.ValidStart
The ID Station has valid start-up data.

ScGainInitial.Ass.ValidConfig
The ID Station has a valid data configuration.

Some of these are environmental requirements

Here is a timing Requirement. Should it be Said better?

Functional Requirements (when combined with assumptions)
Robustness
requirements

Anticipating
Requirements
Validation and Design

- Opening the door to locked.
- Opening the door to unlocked.
- Door opening
- Door closing
- Writing data to the display.
- Validation of any certificate (possibly multiple failures, but at least one success)
- Creation or modification of signed Authorization certificate
- Comparison of fingerprint image and template (possibly multiple failures, but at least one success)

Failure Conditions

SoGanInit.Fail.ReadCard
The card inserted by the user does not allow all the data to be successfully read, possibly due to being
improperly inserted in the first place, being a faulty card, having the incorrect information on it, or being
removed before all the information has been read. The set of data to be read is at least:

- ID Certificate
- U.A Certificate
- Privilege Certificate
- Fingerprint Template (contained in the U.A. Certificate)

SoGanInit.Fail.Fingerprint
A matching fingerprint has not been read, possibly due to no finger being presented, the fingerprint
reader not sensing a fingerprint, or the fingerprint not successfully read within X seconds of the display
requesting a fingerprint. The user will be requested to log in.

SoGanInit.Fail.WriteCard
The card originally inserted by the User does not allow a new Authorization Certificate to be successfully
written, possibly due to being improperly inserted in the first place, being a faulty card, or not being properly
removed before all the data to be read is at least:

SoGanInit.Fail.DoorProp
Once the door has been opened, it is not allowed to close (it is propped open).

SoGanInit.Fail.Audit
Audit files cannot be successfully written. Result: the door is locked and the system is shutdown.

SoGanInit.Fail.AuditProp
Space for audit files has been exhausted. Result: the oldest audit records are overwritten with the new
audit records, and an alarm is raised to the Guard.

Constraints

SoGanInit.Con.NoOtherWks
No ID station reset or Configuration data changes will be allowed during this scenario.

Remarks

SoGanInit.As.UseAuthCert and SoGanInit.As.PrivAcc
These assumptions are negated in the next scenario. User is denied initial access to Enclave, but we need system
processing to be able to distinguish between them. This will lead to additional system functional specification.

SoGanInit.As.PrivAccCert
Note that by requiring the certificates to be preserved, all their constituent parts, such as the Fingerprint
Template, will be preserved.

Issue

What is the value of “K” in SoGanInit.Fail.Fingerprint?

SoGanInit.MT, SoGanInit.MT.SecPrivAcc and SoGanInit.MT.FailWriteCert
There may be an unacceptable performance implications of meeting these requirements. In order to
ensure that the card inserted by the user has had the Author Certificate written to it, once the Author Certificate is
written the ID Station will need to read all the information off again and compare it with the values originally
read. Otherwise, someone could peek a different card into the card reader while the Fingerprint was
being taken. Without being able to monitor the state of the card reader (to detect the card being
inserted), it is not possible to stop someone from inserting a different card and still allowing the
operator to read as a valid behavior. It is no less secure, it just
means that this Author Certificate will not be useful inside the enclave (unless the user is able to do a similar
switch of cards inside).
5.2.10 Administrator log on

Description
An Administrator logs onto the ID Station by inserting their Token in the Admin Token Reader.

Stimulus
A Token is inserted in the Admin Token Reader.

Assumptions
SuLogOn.ActsAsAdmin.
The ID Station is quiescent (no other access attempts, configuration changes or start-up activities are in progress).
The door is closed and locked.
The card inserted by the Administrator has a valid Authorization Certificate.

Success End-conditions
SuLogOn,SuLogOn
The ID Station is available for use by the Administrator, in that it will respond to the commands allowed to the Administrator as defined by the privileges in the Authorization Certificate read from the Token and the Configuration data held on the ID Station.
SuLogOn,SuLogOnSecure.
The door is closed and locked.
SuLogOn,SuLogOnAudit.
The following events have been recorded in the Audit Log (in any order), and the existing audit records are preserved:
- Log-on by Administrator
- Insertion of card
- Reading data from card (possibly multiple failures, but at least one success)

Failure Conditions
SuLogOn Fail,ReadCard.
The card inserted by the Administrator does not allow all its necessary data to be successfully read, possibly due to being incorrectly inserted in the first place, being a faulty card, having the incorrect information on it, or being removed before all the information has been read. The set of data to be read is as follows:
- Authorization Certificate
SuLogOn Fail,Audit.
Audit Eave cannot be successfully written. Result: the door is locked and the system is shut down.
SuLogOn Fail,AuditPresent.
Space for audit files has been exhausted. Result: the oldest audit records are overwritten with the new audit records, and an alarm is issued to the Guard.

Constraints
SuLogOn,CanInterleave.
No ID Station shutdown or user use will be allowed during this session.

Rationale
SuLogOn,ActsValidAdmin.
Only the Authorization Certificate is checked, because we assume that the purpose of the Authorization Certificate is to control access to the workstations within the enclave, and for these purposes the ID Station acts as a workstation. The ID, I&A and Privilege Certificates will have been used to gain entry to the enclave.

5.2.11 Administrator log off

Description
An Administrator logs off the ID Station.

Stimulus
The Token is removed from the Admin Token Reader.

Assumptions
Success End-conditions

ScLogOff_SucLoggedOff
The ID Station is unavailable for use by anyone at the console; it will respond to no commands typed in at the console.

ScLogOff_SucSecure
The door is closed and locked.

ScLogOff_SucAudit
The following events have been recorded in the Audit Log (in any order), and the existing audit records are preserved:

• Log-off by Administrator

Failure Conditions

ScLogOff_FailAudit
Audit files cannot be successfully written. Result: the Door is locked and the system is shutdown.

ScLogOff_FailAuditPreserve
Space for audit files has been exhausted. Result: the oldest audit records are overwritten with the new audit records, and an alarm is raised to the Guard.

Constraints

None.

There is still a need to coordinate and integrate what these scenarios say
Use of Diagrams to Augment the Use Cases

- Not all of our requirements types were addressed by these use cases
- Some of them are addressed separately from the use cases
- Some diagrams are redundant with use cases (to support cross checking)
- Some diagrams could/should be embedded in some of the use cases

3 Domain Information

This section describes the existing Tokeneer system, and the existing ID Station. This allows the proposed variant system structure to be understood in context (see section 4).

The ID Station is part of the larger Tokeneer system, as depicted below. Analysis of the interactions within this larger system has not been carried out, and would normally be done to ensure that a full understanding of the system’s true requirements is obtained. Within the scope of this project, and given that the functionality of the system is already well-defined, this step will not be carried out.
4 System Context

This section defines the system boundary for the high-integrity variant of the ID Station, and explains which elements will be developed by whom, and by what process.

Taking the existing ID Station model given in section 3, we can re-present it showing how the elements will be re-developed.

Environmental requirements
Artifacts are needed in order to specify functional/behavioral requirements

3.2 Biometrics
All of the complexity of the biometrics is hidden within the biometric library, which we will be simulating in a very simple way, enabling the test drivers to determine whether a fingerprint will or will not match a template.

3.3 Door/Latch
The door has four possible states: the cross product of open/closed and locked/unlocked.
Open means the door does not prevent a human from entering or leaving the enclave.
Closed means the door prevents a human from entering or leaving the enclave. To enter or leave, the door must first be opened.
Each type of certificate potentially expands on these attributes.

![Hierarchy of certificate types](image)

**Figure 2.1: Hierarchy of certificate types**

The ID certificate is an X.509 certificate. ID certificates are used during enrolment as we present on tokens.

---

**Augments Functional Rqmt.**

![Diagram of Open/Closed and Locked/Unlocked relationships](image)
As unlocked states are potentially insecure, there is always a time-out period, after which the door will be commanded to lock.

Security Requirement?
Figure 7.2: Administrator logon/logoff state transitions

The context for administrator login is given below.

6 Design Constraints

The system will be developed and run on a workstation running NT 4.0.
(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 no one 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

---

# Approaches to these problems

- Complement requirements with test planning
- Use of requirements processes