XIV. The Requirements Specification Document (RSD)

What is a RSD?
What to include/not include in a RSD?
Attributes of a Well-Written RSD
Organization of a RSD
Sample Table of Contents
An Example

Acknowledgment: these slides are based on Prof. John Mylopoulos slides which are used to teach a similar course in the University of Toronto – St. George campus. Used with Permission.

The Requirements Specification Document (RSD)

- This is the document that is generated by the requirements engineering process; the document describes all requirements for the system under design and is intended for several purposes:
- ***Communication*** among customers, users and designers -- specification should be quite specific about what the system will look like externally
- ***Supporting*** system testing, verification and validation activities -- specification should include sufficient information so that when the system is delivered, it is possible to make sure that it meets requirements
- ***Controlling*** system evolution -- maintenance, extensions and enhancements to system should be consistent with requirements, else the requirements themselves must evolve

Contents of a RSD

- What to include in a RSD:
  - ✓ A complete yet concise description of the entire external interface of the system with its environment, including other software, communication ports, hardware and user interfaces
  - ✓ ***Functional*** (or ***behavioural***) ***requirements*** specify what the system does by relating inputs to outputs;
  - ✓ ***Non-Functional requirements*** (also called ***quality*** or ***non-behavioural***) define the attributes of the system as it operates.
- What ***not*** to include in a RSD:
  - ✓ ***Project requirements*** -- because these are development-specific and become irrelevant as soon as the project is over.
  - ✓ ***Designs*** -- because inclusion of designs is irrelevant to end-users and customers and pre-empts the design phase.
  - ✓ ***Quality assurance plans*** -- for example, configuration management plans, verification and validation plans, test plans, quality assurance plans.

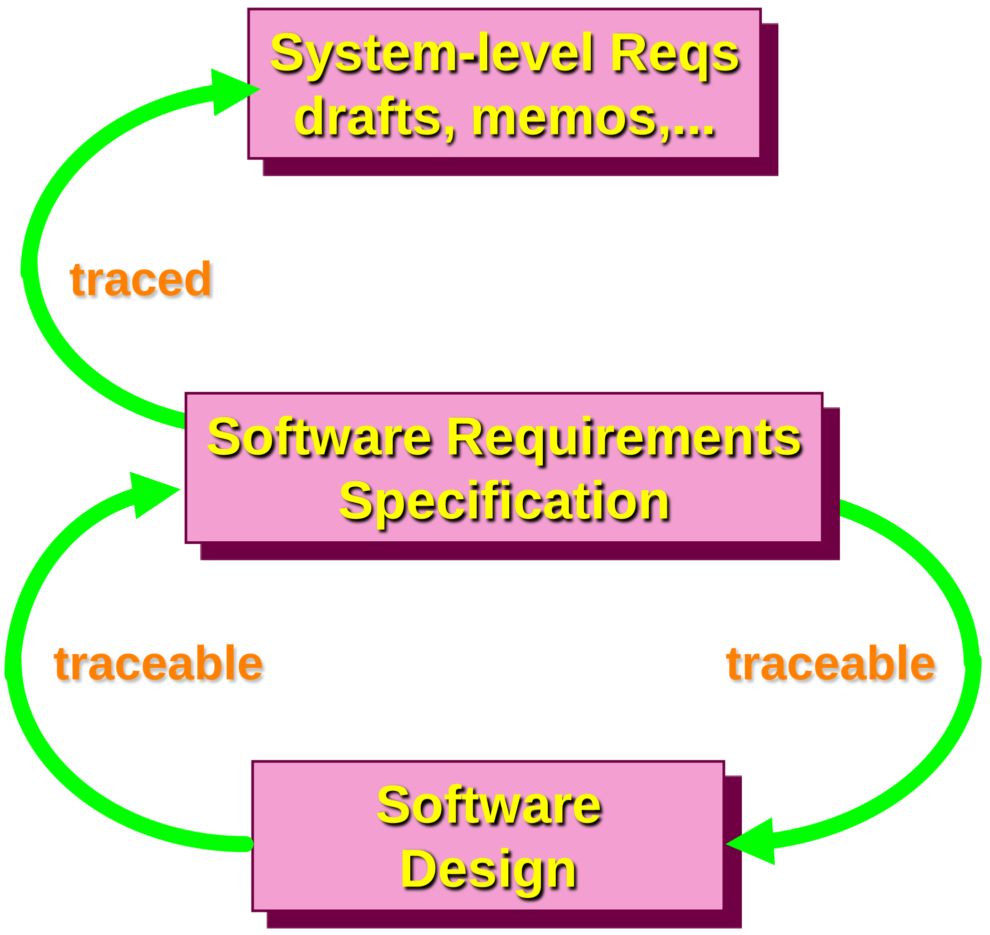
Content Qualities

- ***Correct*** in the sense that all stated requirements represent a need a stakeholder has (customer, user, analyst or designer)
- ***Unambiguous*** in the sense that every stated requirement has a unique interpretation.
- ***Complete*** in the sense that it possesses the following four qualities:
  - ✓ Everything the software is supposed to do is in the RSD;
  - ✓ The response to all possible input combinations is stated explicitly;
  - ✓ Pages and figures are numbered (document completeness);
  - ✓ There are no “to-be-determined” sections in the document.
- ***Verifiable*** in that every requirement can be established through a finite-cost, effective process.
- ***Consistent*** in that no requirement is in conflict with existing documents or with another stated requirement; inconsistencies among requirements may be of four kinds (i) conflicting behaviour, (ii) conflicting terms, (iii) conflicting attributes (iv) temporal inconsistencies

Qualities of a Well-Written RSD

- ***Understandable by customers***, which means that formal notations can only be used as backup to help with consistency and precision, while the RSD document itself is expressed in natural language or some notation the customer is familiar with (e.g., UML.)
- ***Modifiable*** in the sense that it can be easily changed without affecting completeness, consistency; modifiability is enhanced by a table of contents (TOC), an index and cross references where appropriate; redundancy can also be used (mention the same requirement several times, but cross-reference them all)
- ***Traced*** in that the origin of every requirement is clear; this can be achieved by referencing earlier documents (pre-existing documents, drafts, memos,...)
- ***Traceable*** in the sense that attributes of the design can be traced back to requirements and vice versa; also, during testing you want to know which requirement is being tested by which test batch; to enhance traceability (i) number every requirement, (ii) number every part of the RSD hierarchically, all the way down to paragraphs

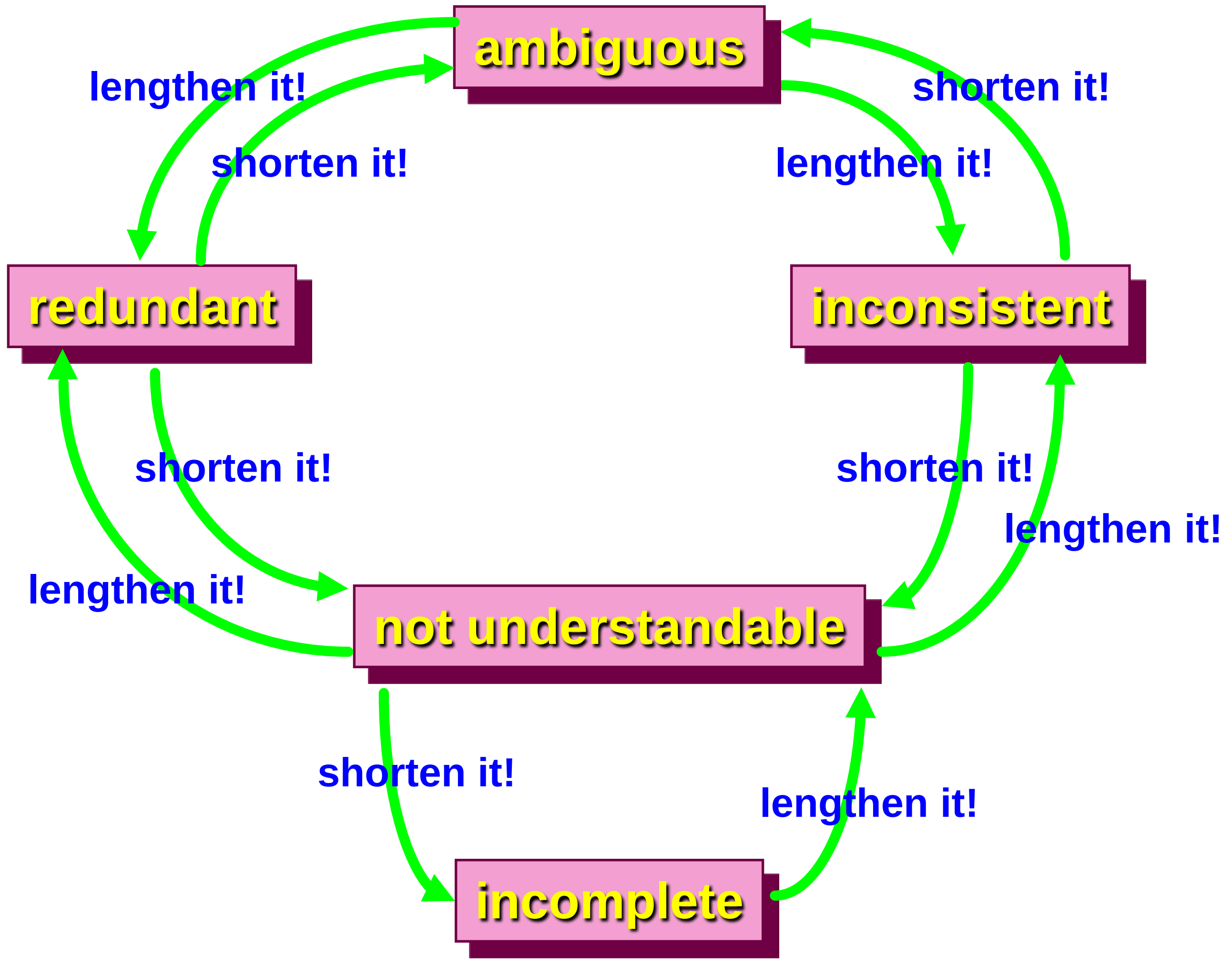
Traceable vs Traced


System-level Reqs drafts, memos,...
traced
Software Requirements Specification
traceable
traceable
Software Design

Style Qualities

- ***Design-independent*** in the sense that it does not imply a particular software architecture or algorithm
- ***Annotated*** in that it provides guidance to the developers; two useful types of annotations are (i) relative necessity, i.e., how necessary is a particular requirement from a stakeholder perspective, (ii) relative stability, i.e., how likely is it that a requirement will change.
- ***Concise*** -- the shorter an RSD document the better.
- ***Organized*** in the sense that it is easy to locate any one requirement.

There is no Perfect RSD!



How to Organize a RSD

- There are many RSD standards, including: US DoD DI-MCCR-80025A, NASA SMAP-DID-P200-SW, IEEE ANSI 830-1984
- Organization may be based on different criteria:
  - ✓ External stimulus or external situation, e.g., for an aircraft landing system, external stimuli or situations might be wind gusts, no fuel,...;
  - ✓ System feature, e.g., call forward, call long distance,...;
  - ✓ System response, e.g., generate pay-cheques;
  - ✓ External object, e.g., by book type for a library information system;
  - ✓ User type.
- It is useful to define a hierarchy among these criteria, use it throughout the RSD document, e.g., sections are defined with respect to (wrt) external stimulus, subsections wrt system feature etc.

Sample Table of Contents

Sample Requirements (cont'd)

5 Data Requirements -- format of data passed through each interface

6 Performance Requirements
6.1 Sizing Requirements
6.2 Timing Requirements

7 Operating Requirements
7.1 Security Requirements
7.2 Safety Requirements
7.3 Restart Requirements
7.4 Backup Requirements
7.5 Fallback Requirements

Sample Requirements (cont'd)

8 Platform Requirements
8.1 Memory Requirements
8.2 Disk Space Requirements
8.3 Operating System Requirements
8.4 Window System Requirements
8.5 CPU Requirements
8.6 Peripheral Requirements
8.7 Network Requirements

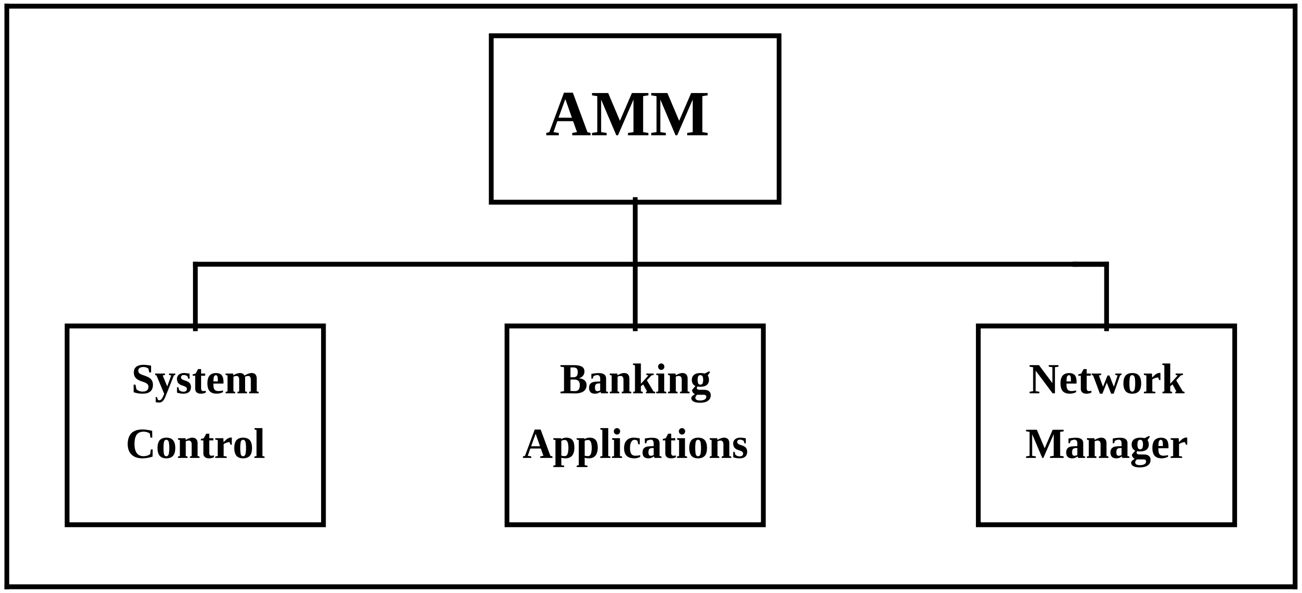
9 Requirements Traceability

Appendices
A.1 Hardware Requirements
B.1 Data Dictionary Notation

Glossary of Terms

Example System Decomposition

An Automated Money Machine (AMM) might be decomposed as follows:

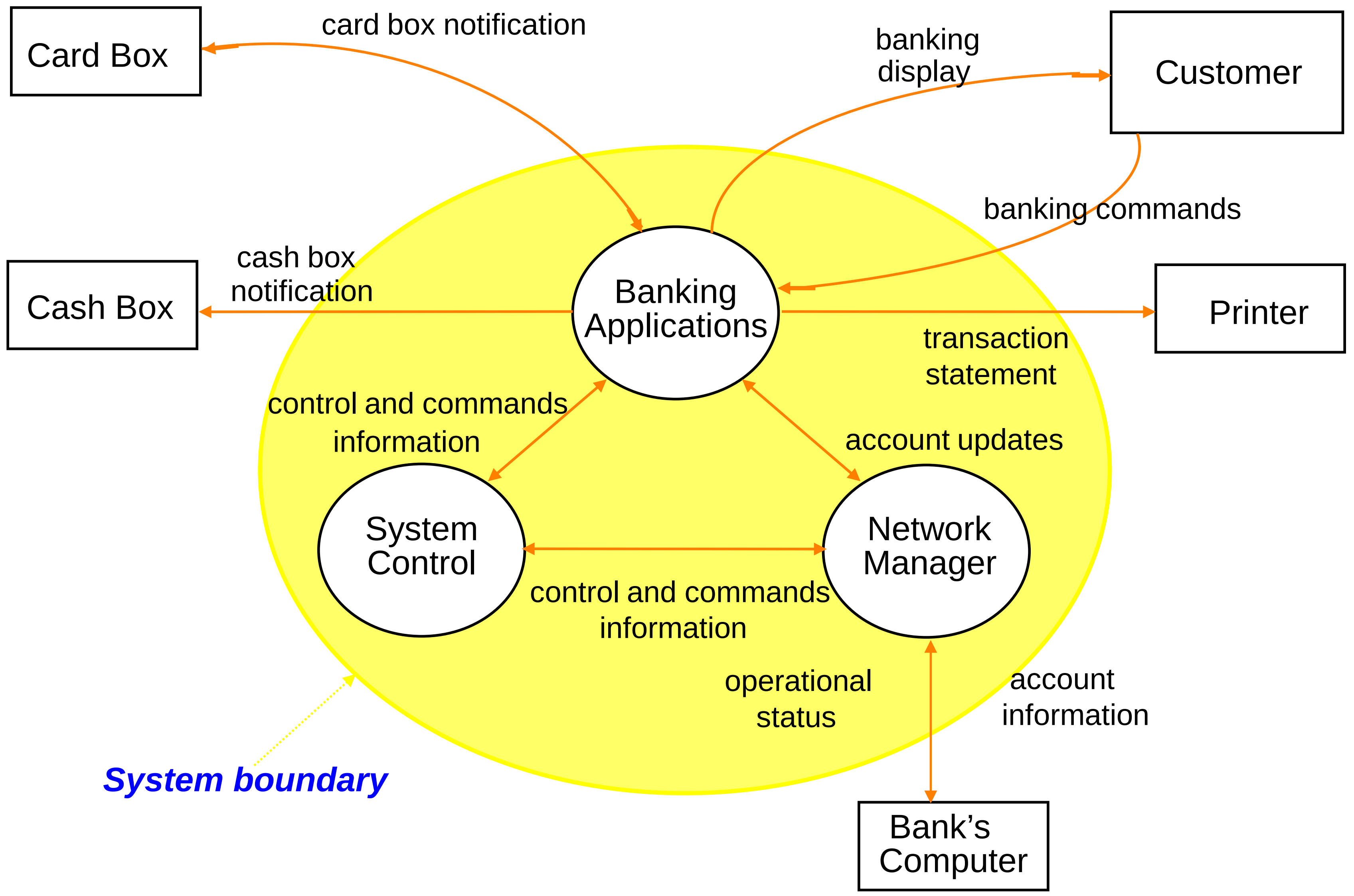


Banking Applications Subsystem includes use cases for banking transactions and their inputs and outputs.

Network Manager Subsystem provides for communication with the central computer system.

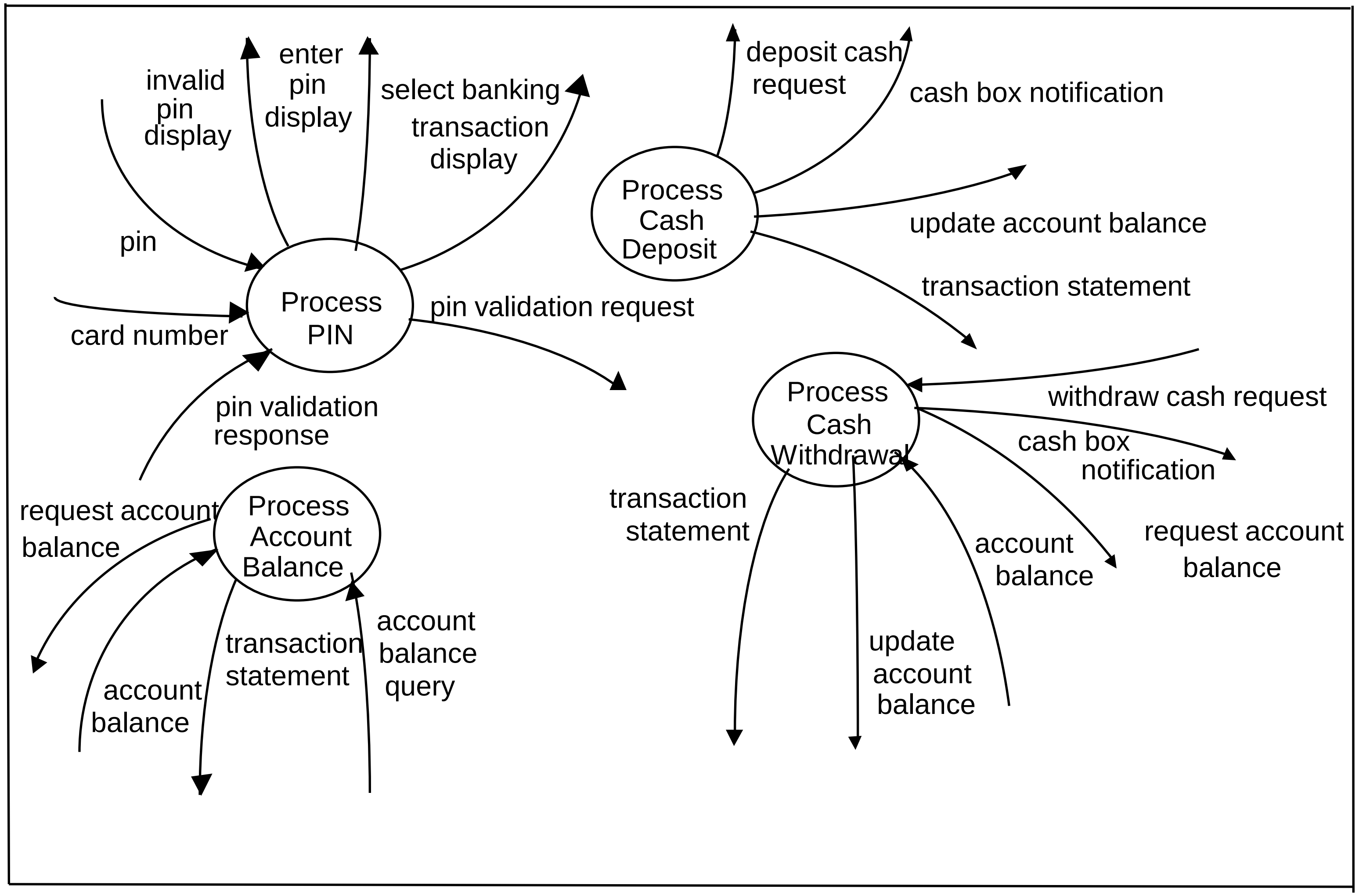
System Control Subsystem is responsible for startup/shutdown control of the AMM system and error handling.

Interfaces


Card Box
card box notification
banking display
Customer
banking commands
Cash Box
cash box notification
Banking Applications
Printer
transaction statement
control and commands information
account updates
System Control
Network Manager
control and commands information
operational status
account information
System boundary
Bank's Computer

Software Requirements for a Subsystem

Define inputs/outputs for each use case for the Banking Applications subsystem



Data Requirements

Name:	enter_pin_display
Description:	Prompt customer to enter his pin.
Composition:	source
	+ destination
	+ message_number
	1{alphanumeric character}80
Name:	error_message
Description:	Software, hardware, or security failure message.
Composition:	source
	+ destination
	+ message_number
	+ error_code
	+ severity_level
	+ error_message_text
Name:	invalid_pin_display
Description:	Inform customer that pin entered was invalid.
Composition:	source
	+ destination
	+ message_number
	+ 1{alphanumeric character}80

References

- [Davis93] Davis, A., *Software Requirements*, Prentice-Hall, 1993, (chapter 3)
- [Thayer90] Dorfman, M. and Thayer, R. *Standards, Guidelines and Examples on System and Software Requirements Engineering*, IEEE Computer Society Press, 1990.