Title Page. The document’s title page should include, at a minimum, the document title, the project name and the team’s name.

Table of Contents. The table of contents should follow the title page. The numbers, titles and page numbers for each section and first level subsections, as well as the letters and titles for each appendix should be included. Second and lower level subsections may be included when reference is needed to provide the reader with direct reference to significant content.

Document Footer. The footer specification for each document should include the document title and page number.

The Table of Contents should follow the title page and the INTRODUCTION (Section 1) should begin on a new page.

The design and layout of each of the CSc 190 and CSc 191 documents should be the same.

1. INTRODUCTION

This is the software requirements specification document for the <name of the project> project sponsored by <name of sponsor>.

This project is being undertaken by the <name of team> development team. The team is comprised of undergraduate students majoring in Computer Science at California State University, Sacramento. The team members are enrolled in a two-semester senior project course required of all undergraduate majors. Successful delivery of the desired software product will fulfill the senior project requirement for the student team members.

PROJECT SPONSOR (if there is more than one person serving as sponsor, list each sponsor):

- Name
- Title
- Company or Organization name
- Contact information (phone number and Email address)

<NAME OF TEAM> DEVELOPMENT TEAM

- List of team member names
- Team contact information (phone number and Email address)

The remainder of this section is intended to introduce the reader to the document and to serve as a high level, executive-like summary to the document.
NOTE. Write this section last! In preparing the document, the project team may have a general understanding of the purpose and scope of the document but one that is not sufficient to inform the general reader. In most cases, it is best to leave this section to the very end, or at least reserve final review and revision until a draft of the document has been completed.

1.1. Purpose

This subsection should describe the purpose of this document. Explain why this document is being written and what the reader should expect, in general, to learn.

1.2. Scope

This subsection should describe what specifically is to be covered by the document and what is not. The reader should be informed as to the context of the information being provided in the document.

1.3. Definitions, Acronyms and Abbreviations

This subsection serves as a glossary for the document. All technical terms used as well as all acronyms and abbreviations used are arranged in alphabetical order. The purpose of this subsection is to provide the reader with quick access to the technical references used throughout the document.

For example:

1.3.1. Definitions

Baseline. A baseline is a work product that has been formally reviewed and accepted by the involved parties. A baseline is changed only through formal configuration management procedures.

Software Requirements Specification. Documentation of the essential requirements (functions/features/uses, performance, design constraints, and attributes) of the software and its external interfaces.

Protocol. A set of conventions that govern the interaction of processes, devices, and other components within a system.

Task. The smallest unit of work subject to management accountability.

1.3.2. Acronyms (definition and/or explanations may need to be included for those acronyms that are not commonly used or understood)

ASP. Active Server Pages *(would require a definition)*

ERD. Entity Relationship Diagram *(would require a definition)*

SSP. Software System Proposal *(would not require a definition)*

XML. Extensible Markup Language *(would require a definition)*
NOTE. You will need to indicate with the appropriate symbol whether the products you reference are “trademarked” (using ® or ™) or “copyrighted” (using ©).

1.3.3. Abbreviations (again, definition and/or explanations may need to be included for those acronyms that are not commonly used or understood)

MGT. Management
LRN. Learning

1.4. References. The following documents and books were used in the preparation of this document.


NOTE. This subsection serves the same purpose as a bibliography. If any documents were used in the preparation of the document, the formal bibliographic information should be specified here. A reference to a specific bibliographic entry should be made at the location in the document where information from that specific source is used.

Example of references used in the preparation of this document:


1.5. Overview of Contents of Document. This subsection briefly describes each of the remaining sections in the document as well as the contents of each appendix. The words used can be identical to those used to introduce each of the sections and each appendix.
2. GENERAL DESCRIPTION

This section contains information that, hopefully, makes the specific requirements (found in the next section) easier to understand. The information provided describes the users and the features that the product is to provide. In addition, this section should include 1) any constraints that will limit the project team’s options in designing the software, and 2) any assumptions or dependencies that might affect the requirements as specified in this document.

2.1 Product Perspective. This subsection describes how the software being developed fits into other products and/or systems in the sponsor’s environment.

If the software is independent and totally self-contained, indicate that this is the case in this subsection.

If the software is to interact with components in a larger system, describe the following:

- The function performed by each component in the larger system
- The interface with these components
- Whatever hardware is to be used by the software

2.2 Use Case Models of the system’s features. Begin with identifying all users that will interact with the system. In Use Case jargon, an Actor/User will interact with the system in order to accomplish the work associated with each feature. The set of features, as identified in the Project Charter, define specifically what services the software system is intended to provide.

In collaboration with the project sponsor, wireframe models should be used in documenting the steps the Actor/User will follow in accomplishing the work associated with each feature.

To find the system’s Actors/Users, begin by asking the following:

- Who will use the system?
- Who will install the system?
- Who will start the system?
- Who will maintain the system?
- Who will shut down the system?
- What other systems will use the system?
- Who will get the information from the system?
- Who will provide information to the system?
- Will anything happen automatically at a preset time?

As indicated above, each feature must identify the specific work to be provided to the Actor/User by the system. The identification of the specific work requirements associated with each feature provides the functionality required to implement each feature. These

---

1 An example of a feature in a University’s Student Administration System is one that provides students with the capability to register for classes. Another example is airline websites that provide a number of features for their customers. One such feature allows users to “book” a flight from one city to another. In both cases significant functionality would be required to implement these features.
functions should be specified with the set of Use Cases needed to provide the appropriate functionality needed to achieve the discrete goals associated with each feature. To help identify the required functionality, begin with be answering the following questions:

- What functions are needed to support each feature?
- Does the feature require access to stored information? What features will create, read, update, or delete (CRUD) that information?
- Does the system need to notify an user about changes in its internal state of the system.
- Are there any external events the system must know about? What functionality is needed to inform the user about those events?

Name and briefly describe each Use Case represented in the system’s Use Case Model. These names and descriptions should be used in section 3 to introduce the specific requirements provided for each Use Case.
The following is an example of a Use Case Model for an Order Processing System [Schneider and Winters, 1998]. “Place Order”, “Return Product”, “Cancel Order”, “Get Status on Order”, “Send Catalog”, and “Register Complaint” would be considered the major features. The other user case bubbles represent the functionality required of each feature. The “Inventory” and “Accounting” systems contains the stored data needed for the application to support all the features. The model’s functional specifications contain the standard create, retrieve, update and delete operations for the data that is stored.
“Manage Orders” and “System Access” are actually use cases identified after analysis and reorganization of the functional use cases specified in the previous examples. “Manage Orders” specifically is the macro use case that contains all the lower level use cases that require order managing. The “System Access” use case provides for the access of the users to the system.

Each of the use cases identified in this subsystem model should be described in detail in section 3 below.

2.3 Interaction Design Specifications for each feature. This subsection includes the set of wireframe blueprints for each feature. These blueprints represent the logical representation of the uses of each feature by the identified Actor/User. The wireframes provide a visual guide that would specify the following detail information:

- Website Navigation
- Content Category and Organization
- Content Layout and Structure
- Call-to-action Items
- Functionality Items
- User Interaction Items
- Methods of displaying the content
- Steps, Methods and Processes for call-to-action, functionality and user interaction items / elements
NOTES:

1. Tutorials and examples are available on the Web (e.g. Google: wireframe examples).
2. Upon the sponsor’s approval the SRS, these wireframe representations as the baseline interaction design requirements for each of the system’s features. Baseline changes to these requirements may be needed during the design and implementation work in CSc 191. The final Actor/User interaction design specification will be represented in the Software Design Specification (SDS) document with screenshots.

2.4 User Characteristics. This subsection contains a description the work that users will expect to be able to do in using the software. In the case where the sponsor will not be the user or will only represent one user type, identifying who the users will be and the work they expect the software to facilitate is essential. Users will be the source of information that will help in identifying the major features that are to be provided. Furthermore, an understanding of the user’s work will guide the design of the user’s interaction with the software and consequently the behavior the software will require of the user.

2.4 General Constraints. This subsection contains any other constraints that will limit the project team’s options in designing the software. Such constraints could include:

- Environmental (or operational) limitations
- Hardware limitations
- Interfaces to other applications
- Parallel operations
- GUI requirements
- Audit and tracing requirements
- Compliance with certain specified standards
- Language requirements
- Protocols
- Security

2.5 Assumptions and Dependencies. This subsection identifies all assumptions and dependencies that the development team expects to affect the requirements as stated in this document. For example, an assumption might be that a specific operating system would be available on the hardware designated to host the software. If, in fact, the operating system were not available, the SRS would then have to be changed.

Note. For purposes of this document, 1) an assumption is something that is taken - or assumed - to be true because the project team is unable to determine the actual truth, and 2) a dependency is a condition that is necessary for the software to be built according to the specifications.

3. SPECIFIC REQUIREMENTS

This section is the most important part of the SRS. Its content contains all of the technical information and data needed to design the software.

The requirements defined in this section should satisfy the following criteria:
• Complete. No requirements should be overlooked. This means that the SRS should include not just the functional specifications. For example, the SRS may need to include performance specifications, design constraints, external interfaces, etc. In addition, the SRS should identify how the software is to respond to all classes of input data and in all realizable situations. Response to valid uses as well as responses to invalid uses must be specified.

• Consistent. No set of individual requirements should conflict with any other set.

• Correct. No errors should exist that will affect the design.

• Modifiable. All necessary changes can be made completely and consistently. This means that each requirement appears in one and only one place in the document and is easily identifiable by section and subsection number.

• Precise. Each requirement is stated unambiguously.

• Measurable and Verifiable. Each requirement is stated in such a way as to allow for the design of a cost-effective process to verify that the software meets the requirement.

• Traceable. Requirements are organized to facilitate the traceability from the SRS, to design, to the implemented software.

NOTE. Section 3 provides for the complete set of baseline requirements. The work to be done in CSc 191 will include the specification of test cases demonstrating that each of these requirements has been satisfied. These test cases will be published in the System Test Specification (STS) Document and be used to verify that the software satisfies the baseline requirements. The results of this testing will be published in the System Test Report (STR) Document.

3.1 Use Case Specifications. This subsection documents each of the Use Cases identified in the system Use Case Model in section 2. Begin by summarizing (listing) the Use Cases by name and in the order in which they will appear below. The Use Cases would appear in the sections numbered 3.1.1 through 3.1.n, where n is the number of Use Cases.

Appendix B contains an example of a use case specification using a standard template. The template include the following:

1. USE CASE ID, NAME and DESCRIPTION with the description being no more than a paragraph.
2. ACTOR(s) NAME, including all Actors who communicate with this Use Case
3. PRIORITY, identifying how important this Use Case is to the project. This is the means of differentiating between “necessary” and “nice to have” features.
4. PRECONDITIONS, which list all the conditions that must be true before the Use Case starts.
5. POSTCONDITONS, which list all the conditions that must be true when the Use Case ends.
6. EXTENDED USE CASES, which is the list of all other optional Use Cases that support this Use Case.
7. INCLUDED USE CASES, which is the list of all other Use Cases used by this Use Case.
8. NORMAL COURSE, the PRIMARY SCENARIO, which is the basic set of events, which describe how the Use Case provides the Actor/User with the indicated use.
9. ALTERNATIVE COURSES, the SECONDARY SCENARIO(s), which provide the alternatives or exceptions to the flow of events described in the Primary Scenario.
10. OTHER REQUIREMENTS some of which are indicated by the template example in Appendix B. The intent is to include any other requirements that affect this Use Case.

3.1.n+1 INFORMATION MODEL. This subsection contains the specifications for the database to be developed as part of the software system. The specification should include an entity relationship diagram (ERD) along with a matrix identifying which relationships are needed to support which use cases.

A byproduct of the functional specification process is the identification of data tables that are to be created and used by the system to support the identified uses. To this point the data tables themselves have not been analyzed for content and organization. This task involves the organization of system data into tables that are simplest in structure and minimal in content.

Two criteria can be applied to specifications of the system’s tables to determine whether their components and organization are optimum for the system they are to support. The first is simplicity. Only a primary key should reference the logical "entry" in a table, and there should be no repeating groups of data within the table. The second criterion is non-redundancy. Redundancy occurs, in part, when the same data element exists within two or more tables. Elimination of redundancy will result in a design where tables are simplified and fewer data elements are needed to be processed and updated.

The procedure used to simplify the initial set of tables is NORMALIZATION. The procedure begins with a set of tables, which were originally identified to support the Use Cases. In the following example, the Class-List data table is specified as the table to be used to produce class lists periodically over the course of a semester. This example will be used to demonstrate the normalization process and the subsequent construction of logical table requirements in the form of the INFORMATION MODEL.

**Data Table:** CLASS-LIST
Composition (Elements and/or Data Structures):

```
Class Number +  // bold indicates primary key element
Class Title +
Class Instructor +
Class Units +
{ Student Name +
  Student ID Number +
  Student Class Level +
  Student Degree +
  Student Major
}
```

Repeating Group
After normalizing this table, the resulting logical table specification is as follows:

**Data Table: CLASS**

Composition (Elements and/or Data Structures):

- **Class Number** +  // bold indicates primary key element
- Class Title +
- Class Instructor +
- Class Units +

**Data Table: CLASS/STUDENT**

Composition (Elements and/or Data Structures):

- **Student ID Number** +  // concatenated key represents the
  relationship between the CLASS and STUDENT data stores
- **Class Number** +

**Data Table: STUDENT**

Composition (Elements and/or Data Structures):

- **Student ID Number** +  // bold indicates primary key element
- Student Name +
- Student Class Level +
- Student Degree +
- Student Major

The resulting information model defines the non-redundant set of data elements, organized for simple access that supports the processing defined in the functional requirements specification.

For each CLASS there are one or more entries in the CLASS/STUDENT table. Each of these entries provides access to the STUDENT table using the Student ID. Clearly, the production of a class list can be supported with this structure. However, note that there is additional information implicit in this representation. For each STUDENT there are one or more entries in the CLASS/STUDENT table. Each of these entries provides access to the CLASS table using Class Number. The production of the student's registration confirmation is therefore possible. If the functional specifications did not include the need for such information, the analyst can indicate to the project sponsor/user that this is possible.

Notation. The following ERD represents the two relationships between the CLASS and STUDENT entities. CLASS/STUDENT is an associative entity type containing the associated Class ID and Student ID key values. For the class list, each Class ID in the CLASS table has one or more Class ID and Student ID entries in the CLASS/STUDENT table. Each of the Student IDs connects to the appropriate student entry in the STUDENT table.
Similarly, for the class conformation, each Student ID in the STUDENT table has one or more Student ID and Class ID entries in the CLASS/STUDENT table. Each of the Class IDs connects to the appropriate class entry in the CLASS table.

There are two cardinality relationships between any two entities. If there are two entities, say Entity A and Entity B, cardinality is a way of specifying how an instance of an Entity A may participate in relationships with instances of the Entity B. There is also a cardinality relationship between instances of Entity B and Entity A.

Cardinality is expressed in terms of a minimum and maximum. For example, an instance of Entity A may be related to one or more instances of Entity B. This would be a "One to many" relationship. In the case where an instance of Entity A may not have any associated instance of Entity B or at most one, that relationship is a "Zero to one" relationship.

The following table shows the four possible cardinality relationships (Minimum to maximum) and the associated notation.

<table>
<thead>
<tr>
<th>Cardinality</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero to one</td>
<td>![Diagram]</td>
</tr>
<tr>
<td>Zero to many</td>
<td>![Diagram]</td>
</tr>
<tr>
<td>One to one</td>
<td>![Diagram]</td>
</tr>
<tr>
<td>One to many</td>
<td>![Diagram]</td>
</tr>
</tbody>
</table>

For a decent explanation of how to build the information model see Ruble [1.4.4].

3.2 Performance Requirements. (this section follows the n\textsuperscript{th}, and last, Use Case specification). This subsection should specify the operational requirements placed on both the software and the users of the software. Development and implementation may occur in a small version of what ultimately will become the software's operational environment. This subsection should make clear the scale in which the software will operate.

The fixed performance requirements might include:
- Number of workstations (clients and servers) to be supported
- Number of simultaneous users to be supported
- Number of data stores to be handled
- Number and size of the attributes associated with each data store
- Number of records to be handled for each data store
The variable (or dynamic) performance requirements would deal primarily with whatever load capacity is expected. For example, the amount of data to be processed within a certain time period should be specified for each transaction type. These specifications might be better described by separating time periods into normal and peek workload categories.

3.3 Design Constraints. This subsection should specify the design constraints that can be imposed by standards and hardware limitations. For example, the sponsor may have existing standards for report formats, data naming, coding, documentation, etc.

In the case of hardware limitations, the project team may have to develop the software to operate with unique hardware configuration. For example, there may be limits on the number of ports, primary (RAM) and secondary (hard disk) memory.

3.4 Quality Attributes. This subsection contains the non-functional requirements, that is, requirements that apply to the software system being developed and not just to a single feature, function or use.

3.4.1 Reliability. The ability of a system or component to perform its required functions under stated conditions for a specified period of time. If practical, reliability requirements should specify the acceptable mean time between failures of the software, averaged over a significant time period.

3.4.2 Maintainability. The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment. This requires both clear documentation of the design specifications, the identification a relationship of the functionality and data base requirements associated with each feature as well as both the documentation and use of a common set coding standards.

Design and construction standards should facilitate the work in modifying the software once the product is delivered and the project team is no longer available. In addition, future work on the software should be significantly shorten the time needed to repair defects that occur.

3.4.3 Program Quality Attributes. The use of the word program refers to a software element, ranging from a module to the entire software product. In this context, a good program has certain measurable attributes, for example:

- **Readability, understandability, and comprehensibility.** These attributes refer to the ease of reading both the documentation as well as the code used to implement the interaction and design of the software systems’ functional components. In the world of real software development and real software, incorporating these attributes into the development process is often neglected. These quality attributes are the most important attributes associated with maintenance of the useful life of the software.

- **Robustness.** This attribute refers to how well the program can handle incorrect input data. A software product should be protected against misuse and be designed to deal with bad input data.

- **Speed of execution.** This attribute refers to how fast the program executes. The speed depends on two major factors, the hardware and the program itself. For a
program to be CPU efficient requires fast and efficient algorithms, full use of available hardware capabilities and appropriate file and data structures and access methods.

3.4.4 Security. The requirements for securing the system against threats to confidentiality, integrity and availability. The level and frequency of access allowed to authorized users should be specified. If prevention against unauthorized use is required, the type of unauthorized user should be described. The level of physical protection of the computer of the computer facility should also be specified. For example, the system might be required to protect against the following:

- Accidental destruction of the software
- Accidental loss of data
- Unauthorized use of the software
- Computer viruses

3.4.5 Transferability / Conversion. Such requirements should specify the ease of moving the software from one platform to another, other than the target system.

3.4.6 Operational Quality Attributes. Quality attributes should be considered for the following software aspects:

- Ergonomic aspects
- Screen layouts
- User interface languages
- Help system

3.4.7 Operations. This subsection allows for the specification of what might be considered as normal and special operational conditions. Some examples are the following:

- unique modes of operation (administrative, data entry, backup and recovery, special categories of use, etc.)
- periods of interactive operation and periods unattended (time driven) operation

3.4.8 Site Adaptations. This subsection allows for the specification of whatever system initialization requirements are necessary. This would include whatever unique modifications needed to install and operate the software on a particular platform.

4. APPROVALS.

This section contains the list of the key signatories necessary to sign-off on the SRS, thereby agreeing to the scope and content of the requirements specified within the document.

More specifically, the approval indicates agreement by the sponsor and the project team that this document represents a shared understanding that the requirements for the project and the system described will satisfy the sponsor’s needs. Any future changes in this baseline specification will be made through the project’s defined change process. The sponsor realizes that approved changes might require the renegotiation of the scope of the work that can be completed within the allotted time.
Approval also constitutes an agreement between the sponsor and the development team that the requirements, as stated, provide all that is necessary for the development team to proceed with the design and implementation of the software system.

Each signatory should be identified by name, title and affiliation. A signature line should be provided along with a date line.

APPENDIX A - Data Dictionary

Three types of data are defined in the data dictionary - data elements, data structures, and data tables. Data structures and data tables are composite definitions, containing other data structures or data elements.

Data elements are listed alphabetically by name. Each data element should contain, at a minimum, the following information:

- Description (1-2 sentences; good example: HomePhone - "residence telephone of this employee". Bad example: Phone - problem is that the element name does not distinguish a residence from work from cell phone number)
- How it is set. (For example, value defined by 1) user input or 2) application calculation, or 3) database default.)
- How it is used. (For example, places where the application references this value or uses it).
- Data structures in which it is included. (For example, FirstName is used in the CustomerName data structure).
- Value type (For example, Date, Character(30), etc.)
- List of discrete values, where appropriate.
- Validity checking rules, where appropriate. (For example, data entry fields that are limited to a specific range of values or a set of values)

Data structures should be listed separately and alphabetically. Each data structure should contain, at a minimum, the following information:

- Description
- How it is set
- How it is used
- The data structure(s) or data store(s) in which it is included
- Composition
  - list the data elements and/or other data structures
  - indicate where subsets of elements and/or data structures appear as a repeating group
  - indicate where a specific element or data structure maybe optional

Data tables should be listed separately and alphabetically. Each data table should contain, at a minimum, the following information:
• Description
• How it is set
• How it is used
• Number of "records"
• Expected growth in number of "records"
• "Record" size in bytes
• Composition

  – List the data elements and/or other data structures
  – Give the length - in bytes - of each data element and data structure
  – Indicate where subsets of elements and/or data structures appear as a repeating group
  – Indicate where a specific element or data structure maybe optional

APPENDIX B – Use Case Modeling tutorial.

There should be a BASIC PATH through the logic of a use case. There can be, however a series of sub-paths that provide specialized instances of the BASIC USE CASE. For example, if a Use Case was required for the selling of two products, then a single Use Case Selling is preferred to two Use Cases called Selling Product A and Selling Product B. Grouping common paths into a class and using that class as a Use Case will significantly reduce the number of Use Cases and therefore enhance the clarity of the model produced.

The functional requirement must be expressed in the users’ language. Do not include the necessary internal mechanics of the system before you have understood what the system is supposed to do. Once you have established the actors and Use Cases, and then begin to describe them.

GRANULARITY. What is the right level of description? A high level functional description would be functions like “Purchasing and Accounting”. A very low level would be specific tasks like “Find Customer Address” and “Validate Stock Level”.

What is the right level? Think of the definition of the Use Case… “a description of a set of sequences of actions, including variants that a system performs to yield an observable result of value to an actor.” [Booch, Rumbaugh and Jacobson, 1994]. There are four (4) rules:

1. “Behaviorally related” means that the interactions should as a group should be self contained… with no intervening time delays. They represent a single functional response to a user.
2. The Use Case serves a single actor in a single place (although a consequence may be to send output to other actors.
3. “Measurable Value” means that the Use Case must achieve some business goal.
4. The Use Case must leave the system in a stable state… it cannot be left partially done.

Focus on those Use Cases that comprise 80% of the systems and not get entangled in the details of the more unusual “edge” cases.

Again, the focus is on identifying Use Cases that add value to the user. Partitioning should be related to these features. The traditional functional decomposition approach does not focus on
the features that add value, but on the way in which the system is organized by function. A rather obscure point, unless you reflect on your Use Case specification!

Do not get hung up on the mechanics of the User Interface before you have really understood what the business requirements are. Keep the intent of the Use Cases external. The Use Case is a sequence of business steps. Treat the system as a black box. This will avoid premature digressions into designing the human interface before you understand what the system is for.

Prototype screens merely depict the interaction between User and Use Case in terms of information and events.

PROBLEMS AND SOLUTIONS IN USE CASE MODELING:

<table>
<thead>
<tr>
<th>THE PROBLEMS</th>
<th>SOLUTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do not cover all aspects of the system; e.g. necessary algorithms, ad-hoc inquiries, etc.</td>
<td>Apply Use Cases to key interactive operational requirements. Identify other types of requirements separately.</td>
</tr>
<tr>
<td>Use Cases lead to functional analysis if used as sole driver of the analysis. This often occurs where use cases are equated with processes on a data flow diagram, leading to functional decomposition.</td>
<td>Use black box events or process threads. Use Cases are a means of identifying the responsibilities of objects and we should constantly bear in mind the object oriented architecture of the system being developed.</td>
</tr>
<tr>
<td>Difficult to assess completeness without over cluttering.</td>
<td>Concentrate on 80% major requirements. Identify other 20% through incremental development (as you go).</td>
</tr>
<tr>
<td>Problem of overlapping Use Cases causes duplications.</td>
<td>Needs careful management. Identify reusable Use Cases and services.</td>
</tr>
<tr>
<td>Too many Use Cases.</td>
<td>Model batch processes, ad hoc inquiries, general maintenance as single aggregate use cases. Use a project log to record alternative courses.</td>
</tr>
<tr>
<td>Not knowing when to stop.</td>
<td>Apply the 80% rule. Concentrate on interactive operational requirements.</td>
</tr>
<tr>
<td>No means of identifying inter-use case dependencies.</td>
<td>Use Pre- and Post-Conditions.</td>
</tr>
<tr>
<td>Incremental development results in partial variant classes appearing, corresponding to Use Cases.</td>
<td>Incremental base should be complete classes not “bits of classes”.</td>
</tr>
<tr>
<td>Use Cases not always of equal value, causing a planning and estimation problem.</td>
<td>Use Cases need to be prioritized according to business importance.</td>
</tr>
<tr>
<td>Lack of formality.</td>
<td>Introduce standard format for Use Case descriptions.</td>
</tr>
</tbody>
</table>

Use Case Specification Template

<table>
<thead>
<tr>
<th>Use Case ID</th>
<th>UC-1</th>
<th>Use Case Name</th>
<th>Request a Chemical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Created By</td>
<td>Tim</td>
<td>Last Updated By</td>
<td>Janice</td>
</tr>
<tr>
<td>Date Created</td>
<td>12/4/2002</td>
<td>Date Last Updated</td>
<td>12/27/2002</td>
</tr>
<tr>
<td>Actors</td>
<td>Requester</td>
<td>Description</td>
<td>The Requester specifies the desired chemical to request by entering its name or chemical ID number or by importing its structure from a</td>
</tr>
</tbody>
</table>
chemical drawing tool. The system satisfies the request either by offering the Requester a new or used container of the chemical from the chemical stockroom or by letting the Requester create a request to order from an outside vendor.

| Preconditions | 1. User's identity has been authenticated.  
|               | 2. User is authorized to request chemicals.  
|               | 3. Chemical inventory database is online.  |
| Postconditions| 1. Request is stored in the Chemical Tracking System.  
|               | 2. Request was e-mailed to the chemical stockroom or to a Buyer.  |

**Normal Course**

1.0 Request a Chemical from the Chemical stockroom
   1. Requester specifies the desired chemical.
   2. System verifies that the chemical is valid.
   3. System lists containers of the desired chemical that are in the chemical stockroom.
   4. Requester has the option to View Container History for any container.
   5. Requester selects a specific container or asks to place a vendor order (alternative course 1.1).
   6. Requester enters other information to complete the request.
   7. System stores request and e-mails it to chemical stockroom.

**Alternative Courses**

1.1 Request a Chemical from a Vendor (branch after step 5)
   1. Requester searches vendor catalogs for a chemical.
   2. System displays a list of vendors with available container sizes, grades, and prices.
   3. Requester selects a vendor, container size, grade, and number of containers.
   4. Requester enters other Information to complete the request.
   5. System stores request and e-mails it to Buyer.

**Exceptions**

1.O.E.1 Chemical is not valid (at step 2)
   1. System displays message: That chemical does not exist."
   2. System asks Requester whether he wishes to request another chemical or to exit.
      3a. Requester asks to request another chemical.
      4a. System starts Normal Course over.
      3b. Requester asks to exit.
      4b. System terminates use case.

1.O.E.2 Chemical is not commercially available (at step 5)
   1. System displays message: "No vendors for that chemical."
   2. System asks Requester whether he wishes to request another chemical or to exit.
      3a. Requester asks to request another chemical.
      4a. System starts Normal Course over.
      3b. Requester asks to exit.
      4b. System terminates use case.

Includes UC-22 View Container History

Priority High

Frequency of Use Approximately five times per week by each chemist. 100 times per week by each member of chemical stockroom staff.

Business Rules BR-28 Only staff who are authorized by their laboratory managers may request chemicals.

DB Requirements List the data tables and the “relationships” required to support the
**Use Case (the project data dictionary should contain detailed specifications for each data table).**

<table>
<thead>
<tr>
<th>Special Requirements</th>
<th>1. The system must be able to import a chemical structure in the standard encoded form from any of the supported chemical drawing packages.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assumptions</td>
<td>1. Imported chemical structures are assumed to be valid.</td>
</tr>
<tr>
<td>Notes and Issues</td>
<td>1. Tim will find out whether management approval is needed to request a chemical on the Level 1 hazard list. Due date 1/4/03.</td>
</tr>
</tbody>
</table>

Partial description of the "Request a Chemical" use case.