INDUSTRY BEST PRACTICES FOR THE SOFTWARE DEVELOPMENT LIFE CYCLE

FHWA/MT-07-006/8117-25

Final Report

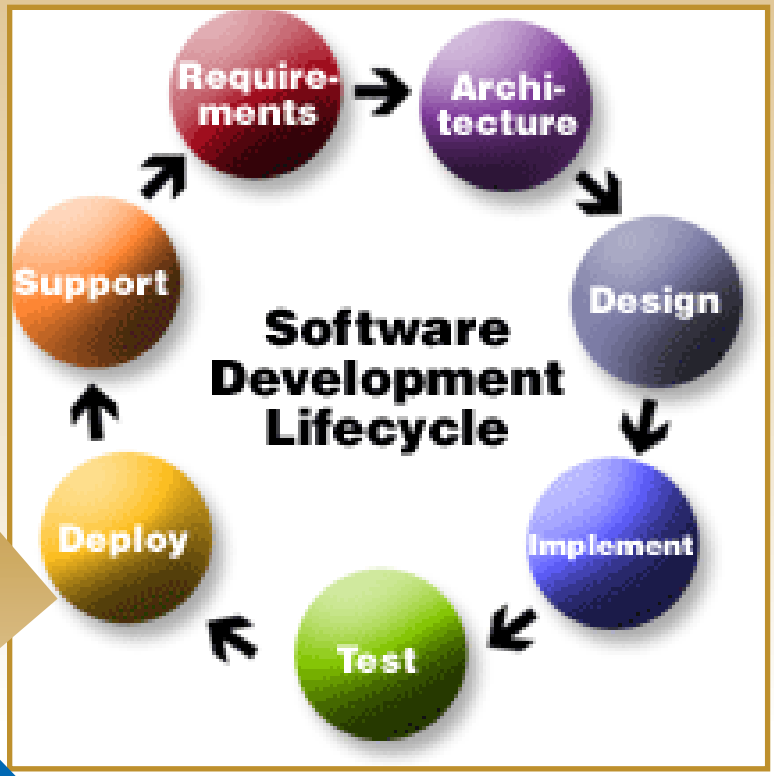
prepared for
THE STATE OF MONTANA
DEPARTMENT OF TRANSPORTATION

in cooperation with
THE U.S. DEPARTMENT OF TRANSPORTATION
FEDERAL HIGHWAY ADMINISTRATION

November 2007

prepared by
Ray Babcock
Gary Harkin
Hunter Lloyd

Montana State University - Bozeman



RESEARCH PROGRAMS

Montana Department of Transportation
serving you with pride

Industry Best Practices for the Software Development Life Cycle

Final Report

for the
Montana Department of Transportation

By

Ray Babcock
Gary Harkin
Hunter Lloyd

Computer Science Department
Montana State University

November, 2007

Technical Report Documentation Page

1. Report No. **FHWA/MT-07-006/8117-25**	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle **Industry Best Practices for the Software Development Life Cycle**	5. Report Date **November 2007** 6. Performing Organization Code	
7. Author(s) **Ray Babcock, Gary Harkin, Hunter Lloyd**	8. Performing Organization Report No.	
9. Performing Organization Name and Address **Computer Science Department Montana State University Bozeman, MT 59717**	10. Work Unit No. 11. Contract or Grant No. **8117-25**	
12. Sponsoring Agency Name and Address **Research Programs Montana Department of Transportation 2701 Prospect Avenue PO Box 201001 Helena MT 59620-1001**	13. Type of Report and Period Covered **Final, 5/2005-11/2006** 14. Sponsoring Agency Code **5401**	

15. Supplementary Notes Research performed in cooperation with the Montana Department of Transportation and the US Department of Transportation, Federal Highway Administration. This report can be found at http://www.mdt.mt.gov/research/docs/research_proj/app_dev/ibp.shtml.

16. Abstract

In the area of software development, there are many different views of what constitutes a best practice. The goal of this project was to identify a set of industry best practice techniques that fit the needs of MDT, and provide a consistent and robust process for software development.

The researchers believe that the Unified Software Development Process represents the closest methodology to an industry standard for software development. The Project Management Professional certification of the Project Management Institute represents best practice in project management. The researchers have suggested the Unified Modeling Language as a representational method for developing software designs. Practical suggestions for the elicitation of software requirements are provided. The complete set of MDT forms was converted to an XML utility system to allow for easy changes and configurations.

17. Key Words **Industry Best Practice, Unified Modeling Language, Software Requirements, Software Design, and Software Development**		18. Distribution Statement **Unrestricted. This document is available through the National Technical Information Service, Springfield, VA 21161.**	
19. Security Classif. (of this report) **Unclassified**	20. Security Classif. (of this page) **Unclassified**	21. No. of Pages **159**	22. Price

Disclaimer Statement

This document is disseminated under the sponsorship of the Montana Department of Transportation and the United States Department of Transportation in the interest of information exchange. The State of Montana and the United States Government assume no liability for its contents or use thereof.

The contents of this report reflect the views of the authors, who are responsible for the facts and accuracy of the data presented herein. The contents do not necessarily reflect the official policies of the Montana Department of Transportation or the United States Department of Transportation.

The State of Montana and the United States Government do not endorse products or manufacturers. Trademarks or manufacturers' names appear herein only because they are considered essential to the object of this document.

This report does not constitute a standard, specification, or regulation.

Alternative Format Statement

MDT attempts to provide accommodations for any known disability that may interfere with a person participating in any service, program, or activity of the Department. Alternative accessible formats of this information will be provided upon request. For further information, call (406) 444-7693, TTY (800) 335-7592, or Montana Relay at 711.

Contents

List of Tables

1 Introduction

The Montana Department of Transportation (MDT) requested industry best practices methods and tools for their Software Development Life Cycle (SDLC). The current methodology was studied and compared to industry practices. This report and associated documents are the result.

An industry best practice is a technique or methodology that, through experience and research, has proven to reliably lead to a desired result. In the area of software development, there are many different views of what constitutes a best practice. As pointed out in the call for proposals by MDT, there are a number of organizations that offer industry best practice methodologies and they are not always compatible nor do they have a common perspective on the goals of a software development effort. The goal of this project was to identify a set of industry best practice techniques that fit the needs of MDT, and provide a consistent and robust process for software development.

A wide variety of current industry methodologies were identified; the research focused on those methodologies that are both tried-and-true and widely accepted. It is the opinion of the researchers that the *Unified Software Development Process* (UML) [Jac1999] represents the closest methodology to an industry standard for software development and the researchers have borrowed heavily from those methods. Also, the researchers believe there is much to be learned from the Agile development methods and have referred to those liberally [McC1996]. The Project Management Professional (PMP) certification of the Project Management Institute (PMI) represents best practices in project management [Hel2004] and that information is used extensively in developing strategies for conducting a software project. The Capability Maturity Model Integration (CMMI) for Software is widely used worldwide for process improvement and reengineering, and those methods are used where appropriate [Ahe2004]. For a complete analysis of software engineering, *Software Engineering* [Pre2001] was used as a guide, but the entire area of software engineering is well developed and virtually any book will do. The Unified Modeling Language is suggested as a representational method for developing software designs, *UML Distilled* [Fow2004] and *UML for Mere Mortals* [Mak2005] are recommended as good user references; *The Unified Modeling Language Reference Manual* [Rum2005] is recommended for a complete analysis. In the tried-and-true category, this document incorporates information from *Joel on Software* [Spo2004], *Facts and Fallacies of Software Engineering* [Gla2003], *The Software Development Edge* [Mar2005], and *Peopleware* [DeM1999]. For user interface design, *Don't Make Me Think* [Kru2000] and *User Interface Design for Programmers* [Spo2001] are recommended.

One issue that is important to discuss is how to deal with multiple viewpoints of industry best practices. The researchers believe that consistency and robustness are more important to the success than having every bell-and-whistle; therefore, only a few methodologies are used and combined in a manner that provides those benefits. As mentioned previously, the *Unified Software Development Process* (USDP) [Jac1999] is the industry standard analog to the *Rational Unified Process* from Rational Systems and is widely used for software development in industry. USDP is a relatively complex

process that focuses on the software development process from inception through release. While it is strong in software development, it is weak in the area of project management. The *Process Management Professional* [Hel2004] strategies have excellent recommendations for project methodologies, but do not consider software development explicitly.

The Capability Maturity Model Integration [Ahe2004] is primarily about process reengineering, which can be a part of every software development project, and provides excellent practices for software project management, but the scope is too narrow to be the only source. Software CMM is the staged model from CMMI that is pertinent to software development; it has five levels with a set of key process areas dictated for each level:

1. Initial, focus is competent people;
2. Repeatable, focus is basic project management;
3. Defined, focus is process standardization;
4. Managed, focus is quantitative management; and
5. Optimizing, focus is continuous process improvement.

While not specifically stated, Level 3 could be identified as the appropriate focus for this project as MDT attempts to standardize its software development process in accordance with industry practices. That level will be the primary focus of this work, along with some process areas from Levels 2 and 4. The key process areas are:

- Statistical Process Management (L4)*,
- Peer Reviews (L3)*,
- Project Interface Coordination (L3),
- Software Product Engineering (L3)*,
- Integrated Software Management (L3)*,
- Organization Training Program (L3)*,
- Organization Process Definition (L3)*,
- Organization Process Focus (L3),
- Software Configuration Management (L2)*,
- Software Quality Assurance (L2)*,
- Software Acquisition Management (L2), and
- Software Project Control (L2)*.

Those marked with an asterisk are impacted by this work. Due to the limitations on the scope of this project, none of these will be completely satisfied, but all will be improved.

Other industry best practices methodologies provide similar methods and characteristics. This report provides a compilation, which is at the cutting-edge technologically, but also verifiably practical and robust. Existing MDT documents and templates were used as much as possible and have changes to be made indicated where appropriate.

1.1 Report Organization

This report is not a single document, but a series of documents and a web site that provide a strategy for implementing an industry best practices software development methodology, including templates, forms and software recommendations. The report consists of five parts:

- Recommendations and discussions of each part of the project as contained in this particular document;
- A web site that details:
    - A series of processes outlining a series of steps to be followed during various phases of the project,
    - Documents to be used in the various phases of the project cross-referenced in the process outlines,
    - Templates and methods to be used in software used to support a project, and
    - Where appropriate, web pages that would be used to support a project.

The division of information follows that there is the SDLC process and several sub-processes that represent the fundamental nature of software development:

- The SDLC,
- Requirements development and management,
- Project management,
- Software design,
- Software development, and
- Project implementation.

There is overlap between these processes in some areas, but this breakdown provides reasonably concise bodies of material with common methods that integrate into the SDLC.

2 The Software Development Life Cycle

The SDLC is the set of phases that a project must complete. The proposed phases for the MDT are Planning, System Analysis, System Design, Technical Design, Development, Implementation, and Conclusion. In various references, there are different breakdowns of the SDLC, but they contain effectively the same steps. Some are software-oriented and have no planning step, many do not differentiate between the System and Technical Design phases, and there are slight differences in the order of the steps or their content. The number of phases in the organization of the tasks is largely irrelevant within the range of those strategies that are proven to work. The researchers believe that MDT proposed phases are workable and meet the needs of MDT. The use of both System and Technical Design phases is advantageous because it breaks a relatively complex phase into two parts, which can be managed more effectively.

The justification for these choices can be found in [Jac1999] which describes the USDP life cycle. It has 5 phases: Requirements, Analysis, Design, Implementation, and Test. MDT adds a planning process, which is necessary in a business context, combines Requirements and Analysis in one phase and breaks the Design phase into two phases. That seems to be a large change, but it actually provides an identical result, pushing part of the Analysis phase into System Design and leaving the System Analysis phase as essentially the same as the USDP Requirements phase. MDT could, theoretically, add a Requirements phase, but there seems to be no good reason for doing so. The MDT process includes Test in Development and Implementation, but that is not a significant difference. The Software CMM requirement for Organization Process Definition at Level 3 [Ahe2004] is met by this process.

Each phase of the SDLC is discussed below in terms of the justification for the tasks in the phase. There is one part of this process that is universal, and that is the Project Planning Process, which is a subsystem that impacts each phase. The Project Planning Process schedules and assigns resources to tasks and details the collection of data to be used in evaluating performance and improving estimates. It will be covered independently in some detail.

The SDLC process is detailed in the project web site.

2.1 Planning

The planning phase is dictated by the MDT IT Planning Process document, which requires the preparation of an IT Project Nomination to be submitted for approval. The proposed Planning Phase process is designed to provide the information for the Nomination and to establish a foundation for continuance of the project. The tasks proposed are shown on the web site and will not be repeated here.

The tasks are based on information in [Ahe2004] and [Hel2004]. The CMMI specifies in Generic Process (GP) 2.2, the need to establish and maintain a plan for performing the process. The PMP mandates that project initiation and planning perform:

- An analysis of need,
- The development of project goals,
- The collection of project requirements,
- A list of project deliverables,
- The identification of project constraints,
- The development of a project schedule, and
- An estimate of projected resource needs and budget.

The proposed list of tasks meets these requirements and honors the specifications of the MDT process. The Use Case Business Model is derived from the USDP and represents the researchers' belief that UML is the best overall design methodology for consistency throughout the life cycle.

2.2 System Analysis

This phase takes the preliminary plan and transforms it into a detailed project plan that can be used to control and monitor the project. The proposed tasks are a combination of suggestions from the USDP and the PMP. The key deliverables from this phase are the requirements that drive the software development and the detailed project plan that drives the lifecycle. Each of these is a dynamic document that is subject to change in succeeding phases, but those changes are an important metric to be collected and used in improving the SDLC process.

The Requirements Document and Project Plan are so important that they are treated separately in this document.

The justification for the task list can be found in the USDP [Jac1999] and in the PMP [Hel2004]. The USDP produces a Requirements Document; the PMP suggests that the phase should produce a Workscope and Baseline Project Plan. Updates for the benefit-cost analysis to provide a check on the initial estimates, team selection, and planning for the next phase have been added.

A risk management plan is mandated by both the USDP and the PMP to insure that potential problems are identified and handled at each stage.

2.3 System Design

This phase converts the requirements and data dictionary into a software architecture, implementation strategy, and produces an updated project plan. The proposed tasks are derived from the USDP [Jac1999] and for the project planning portion, from the PMP. The USDP Analysis phase produces an Analysis Model, which includes a data dictionary that might be separate in a non-object-oriented programming environment. In the USDP, user interface design is in the Design phase, but the researchers believe that best practices would break this into two parts to avoid costly mistakes. This is discussed in more detail later. The need to design Acceptance and System Tests early is addressed here. The project planning portion follows the requirements of the PMP [Hel2004] with regard to continuous updating of the plan. Meeting the CMMI [Ahe2004] requirements for

improved quality processes adds a number of components, including a training plan, test plans, user support plans, and documentation plans.

2.4 Technical Design

This phase converts the system design into a technical design that is more attuned to implementation in a programming language and produces mockups of the user interface design. The proposed tasks are derived from the USDP [Jac1999] and for the project planning portion, from the PMP. The USDP Design phase produces a Design Model, which includes a system design, subsystem designs, interface specifications, and class architectures. A System Specification, which is a more formalized presentation of the Requirements Document, was added to this list. Having such a document reduces the likelihood of potential design errors. Unit tests are designed here, although the USDP suggests that this happen in the Implementation phase. The researchers feel that performing the design here will speed implementation by not having the programming team spending its time on this activity, or possibly ignoring it. The USDP addresses the user interface design issue here as well.

The project planning portion follows the requirements of the PMP [Hel2004] with regard to continuous updating of the plan. Meeting the CMMI [Ahe2004] requirements for improved quality processes adds a defect tracking system to insure that design and implementation errors are identified, their resolution verified, and the whole process measured.

2.5 Development

This implements the system design to produce running software and documentation. This is the Implementation phase in the USDP and fits the task list here, although it is obviously a complex activity. The PMP and CMMI best practices mandate a number of tasks based on preliminary work in earlier phases, but it also creates a configuration plan and management system as mandated by the Software CMM.

2.6 Implementation

This phase performs the installation and acceptance testing of the software and training for users. This phase includes most of the Test phase of the USDP, but it includes significant tasks that attempt to achieve the process quality goals of the CMM software, including the user support plan, training, and execution of a user survey. While the project is nearly complete, it is important to perform final updates on the specification and requirements documents, and the project plan to insure that any changes discovered in testing are properly accommodated and project performance is adequately monitored.

2.7 Conclusion

This phase moves the software into the maintenance cycle, reviews the results of the implementation and the conduct of the project, and makes recommendations for the future. The USDP does not have this phase, but the PMP requires that a project have a closeout stage to provide feedback for further projects. Also, the CMMI model of continuous process improvement mandates that this phase continues indefinitely. It might be worthwhile to change the name of this phase to Maintenance if MDT deems it appropriate. From a project point of view, defect tracking is an important metric to feed back into the system for purposes of better estimating project requirements, and identifying design and development issues that require attention.

3 Requirements Development

The requirements are the driving mechanism for a software development project and one of the most common areas mentioned in project failures. Without well-researched and written requirements, it is impossible to meet the expectations of the users; poor requirements leave a software project with little direction and no criteria for controlling the development process. Unfortunately, good requirements are elusive because they are typically based on non-specific descriptions of needs that are voiced by people who don't understand software, and requirements creep during a project can lead to extended development time and project failure.

This is a problem of requirements development and project management. Here both are discussed, but the focus is on the methods needed to develop good requirements documentation.

3.1 Introduction

Even the smallest software development project benefits from clear requirements. Building a house or other engineering project without plans is inconceivable. However, many development projects begin with a brief interview and go straight to coding. The most fundamental need according to industry best practices for successful software development is to begin with a good set of requirements.

Requirements communicate "what" is to be built. They describe in detail the proposed inputs provided by the users and the proposed outputs generated by the software. Graphical user interfaces are designed based on initial figures from the requirements document.

Most people agree that the problem with developing good requirements is a problem of communication between a developer and a user. Often they don't speak the same language. Failures in communication often show up in software that is difficult if not impossible to use.

Many books have been written and many procedures for developing requirements have been tried by industry. These range from highly structured packages that cost many thousands of dollars down to newer ad hoc loosely structured systems, such as extreme programming. The cheapest and most effective method that works for the largest variety of projects is to simply use English. Unambiguous English to be sure, but just plain pictures and plain English carefully written provide requirements that can be read and understood by both developers and users alike. They provide a written record of what the proposed software development is to create and, when carefully done, provide an unambiguous presentation. More detailed "developer oriented" software specifications can be developed using the software requirements document as a safe and secure starting point.

A good software requirements document is a dynamic, constantly changing, record of what is desired for the software to accomplish. As prototypes are shown to the users and

more people read the requirements, changes can be handled by simply changing the requirements document. At some point the requirements need to be "frozen" where additional changes are held until the next version. However, error corrections and functional modifications discovered as the software development proceeds can and should still be reflected in the current requirements document. Even as the system is put into operation, the requirements document should reflect the current system. If this is done, the requirements document becomes a useful document throughout the entire life cycle of the developed software.

3.2 Requirements Elicitation

Here is where a development team shines. Elicitation of a good set of requirements is fundamental to the success of a software development project. Leffiingwell and Widrig [Lef2003] in *Managing Software Requirements* give a good list of elicitation methods.

These are:

- Using Features,
- Interviewing,
- Requirements Workshops, and
- Brainstorming and Idea Redirection Storyboarding.

However, any sequence of activities that clearly define a set of requirements is considered useful.

3.3 Using Features

System features are high-level expressions of desired system behavior.

In all the elicitation methods described, something should be written down. It does no good to the development team if one user has a clear idea of what should be done, but nothing is written down to substantiate this user's vision.

High-level expressions are usually not well defined, but they can lead a development team to eliciting more specific requirements. They are often related to the user's view of system behavior. However, the development team should not leave these at a high level. They should search for the need underlying the feature described. Features and requirements, for that matter, must address real needs, not just pie-in-the-sky wishes. Another definition of features proposed by [Lef2003] is "a service the system provides to fulfill one or more stakeholder needs".

Some example features are listed below:

- Manual control of doors during fire emergency,
- Provide up-to-date status of all inventoried items,
- Provide trend data to assess product quality,
- Report deductions-to-date by category,
- Vacation settings for extended away periods,

- Minimum of two independent confirmations of attack authorization required, and
- Windows XP compatibility.

3.4 Interviewing

Probably the most useful and most common technique to elicit requirements is the interview. Just sitting around and talking does not work. A more structured approach is required to produce good results.

Choosing the right people to interview is important. Try to have at least one representative from each stakeholder group. These can be done at different times, but everyone needs to be included.

A good technique to use when doing an interview is the recording of a restatement of the comment provided by a stakeholder. This gives the stakeholder the ability to hear what they said repeated and correct any misconceptions on the spot. An example follows:

Developer: "How many different invoices are used in a typical month?"
User Sue: "Fred, how many would you say?"
User Fred: "Oh, I'd say probably 120."
User Sam: "Oh, that's not right, we often do over 200!"
User Sue: "How about we say 250 to be safe?"
User Fred: "Well he didn't ask for a safe number, he asked for a typical number. I'd say 175."
User Sue: "O.K. anyone have a problem with 175?"
Other Users: "No."
User Sue: "We typically use about 175 invoices each month."
Developer turns on his recorder and records "The company uses 175 invoices in a typical month".

This simple technique can provide a record of what is discussed without irrelevant material being recorded. It also can catch a misunderstanding if it occurs. For example, suppose the following is the developer's recorded statement after the above comments:

Developer turns on his recorder and records "The company uses a maximum of 175 invoices in a typical month".
User Sue: "Wait a minute! We said that was typical not maximum."
Developer turns on his recorder and records "Correction the company uses an average of 175 different invoices in a typical month."

After the developers return to their location, the recorded sessions are played for the entire development group and a set of requirement statements are generated.

3.5 Context Free Questions

Another good technique to enhance the success of interviewing in gathering requirements is the use of "context free questions". These are questions that can be asked without regard to the context being discussed. They could be asked about ANY software

development project. Using them often elicits extremely useful information to enhance the quality of the requirements gathered.

Example Context Free Questions follow: (The project name is assumed to be MDTProject1)

- Who is the client for MDTProject1?
- What is a highly successful solution really worth to this client?
- Should we use a single design team or more than one?
- How much time do we have for this project? What is the trade-off between time and value?
- Where else can the solution to this design problem be obtained? Can we copy something that already exists?
- What problems does this system solve?
- What problems could this system create?
- What environment is this system likely to encounter?
- What kind of precision is required or desired in the product?
- Am I asking you too many questions?
- Do my questions seem relevant?
- Are you the right person to answer these questions?
- In order to be sure that we understand each other, I've found that it helps to have things in writing so I can study them at leisure. May I write down your answers and give you a written copy to study and approve?
- Is there anyone else who can give me useful answers?
- Is there some place I can go to see the environment in which this product will be used?
- Is there anything else I should be asking you?
- Is there anything else you want to ask me?

Of course, some of these may not be appropriate for a particular project. It will help if the entire list is scanned for each interview to be sure to include appropriate ones.

(From *Exploring Requirements: Quality Before Design* by Gause & Weinberg [Gau1989].)

3.6 Requirements Workshop

A well run and productive requirements workshop has the potential to be the most important requirements elicitation method.

The first step in a successful workshop is gathering the right people. All stakeholders in the project should be represented. The requirements workshop assists in building an effective team, committed to one common purpose: the success of the project.
The workshop provides a unique opportunity for stakeholders from various parts of the organization to work together toward the common goal of project success.

To prepare for a workshop it is important to sell the concept inside the organization. This can be done by communicating the benefits of the workshop to members of the team. Preparation also consists of ensuring the right people attend and attending to the logistics (structure of invitation, travel arrangements, meeting room selection) with a high degree of professionalism. Lastly, preparation consists of providing good "warm-up" materials. These should be sent out in advance and consist of project-specific information and some "out-of-the-box" thinking preparation. Forget all the preconceived notions of what can't be done and open the door to any possibility.

Another important consideration for the workshop is picking the facilitator. This person will run the workshop and should be someone outside the organization with no stake in the project. It is good to have someone with experience in the problem domain. If, however, you can't find such a person, a team member may substitute if they will agree to the following:

- The facilitator will receive some training in the workshop process.
- The facilitator has demonstrated solid team-building skills.
- The facilitator is personable and well respected by both internal and external team members.
- The facilitator is strong enough to chair what could be a challenging meeting.

If the workshop is to be facilitated by a team member, that team member must not contribute to the ideas and issues at the meeting. It is important to maintain objectivity so as to get at the real facts.

Leffingwell and Widrig [Lef2003] list the following responsibilities of the facilitator:

- Establish a professional and objective tone for the meeting.
- Start and stop the meeting on time.
- Establish and enforce the "rules" for the meeting.
- Introduce the goals and agenda for the meeting.
- Manage the meeting and keep the team "on track".
- Facilitate a process of decision and consensus making, but avoid participating in the content.
- Manage any facilities and logistics issues to ensure that the focus remains on the agenda.
- Make certain that all stakeholders participate and have their input heard.
- Control disruptive or unproductive behavior.

It is suggested that the agenda follow the following rough outline:

- 30 minutes: Introduction (Review agenda, facilities, and rules);
- 90 minutes: Context (Present project status, market needs, results of user interviews, etc.);
- 2 hours: Brainstorming (Brainstorm features of the application);

- 1 hour: Lunch (Work through lunch to maintain momentum);
- 1 hour: Brainstorming (Continue to brainstorm features);
- 1 hour: Feature definition (Write two- or three-sentence definitions for features);
- 1 hour: Idea reduction and prioritization. (Prioritize features); and
- 1 hour: Wrap-up (Summarize and assign action items).

As with any human endeavor, problems may arise. Below are some suggestions for handling some common ones.

Problem	Solution
Time management	Facilitator keeps a kitchen timer to time the meeting and all breaks.
Grandstanding, Domineering positions	Facilitator enforces a "5 minute position statement" rule. Also, create a "parking lot" list for later discussion of ideas that deserve discussion but aren't relevant to the agenda item.
Lack of input	Facilitator makes it clear that no one should leave the workshop without having provided an idea or supported the idea of another.
Negative comments	Facilitator should limit these kinds of comments to one per person without penalty. After that a penalty is assessed for each one.
Flagging energy after lunch	Serve a light lunch & provide a mid-afternoon snack break. Move furniture, change lighting and temperature. Do what is possible to keep the participants awake.

3.7 Brainstorming and Idea Redirection

Brainstorming involves both idea generation and idea reduction. The significant stakeholders gather together in one room and supplies are distributed. These can be as simple as a stack of large sticky notes and a thick black marker for writing on the notes.

The facilitator gives the rules:

- Do not allow criticism or debate.
- Let your imagination soar.
- Generate as many ideas as possible.
- Mutate and combine ideas.

The facilitator may start things off by asking questions similar to the following:

- What features would you like to see in the product?
- What services should the product provide?
- What opportunities are we missing in the product or the market?

The facilitator then asks the participants to share their ideas aloud and to write them down, one per sheet. Speaking aloud can generate a cascade effect of ideas among other participants. No criticism or debate is allowed at this point. As ideas are generated, the facilitator collects them and posts them on a wall in the meeting room. If possible, the facilitator may organize them into common themes, but the first priority is to get them up and visible to all.

Ignore lulls in the production of ideas. Often a space in activity is when everyone is thinking hard and, if given a bit of time, may lead to the best suggestions yet. In our fast paced life, it is often uncomfortable to have quiet reign. But, give it time and the process will begin again in earnest.

After the main initial process, a prioritizing phase takes place. Some kind of voting process should be established to rank the ideas in order of importance. An organization of all ideas into the categories "critical"," important", or "useful" may help. Critical means indispensable. Important means there could be a significant loss of customer utility. Useful means nice to have.

The lists are gathered, written down on paper, and given to the design team for review. Sometimes it may help to have another brainstorming session at a later date, but, the purpose it to elicit as many good requirements as possible in the shortest possible time. The design team can do the detailed feasibility review of the proposed feature.

3.8 Storyboarding

It is said that a picture is worth a thousand words. In gathering requirements, it is often more efficient for two people to discuss a problem looking at a screenshot instead of just staring at each other.

Storyboarding has long been used in the film industry to plan the sequence of events in a movie. A complex software package is at least as complicated as a movie with the added non-linearity of menu selections, active buttons, and hyperlinks.

The purpose of storyboarding is to gain an early reaction from the potential users of a software package. They are easy to construct and give a user a representative view to criticize.

Storyboarding is inexpensive, user friendly, informal, interactive. Users can see for themselves if a proposed interface will do the job from their perspective. If a correction is suggested, this can be done on the storyboard very cheaply and proceed on to the next change with a minimum of delay. If a user is having a problem of "where to begin" in describing what they want, a quick sketch of a user interface can get them off square one.

Today, storyboards can be constructed as passive, active, or interactive. Passive storyboards, similar to the film industry, consist of sketches, pictures, and screen shots. The analyst walks the user through the proposed software by guiding them through the respective storyboards and gauging their responses. Active storyboards are animated or automated. They automatically produce a sequence of images for the user, often with a pre-recorded audio narration. Interactive storyboards allow the user to "play" with a simulated interface. Often constructed as rapid prototypes of the proposed software package, an interactive storyboard gives the user the actual look and feel of the proposed interface.

The detail level and type of storyboard will depend on the clarity that exists for the project. A rather small project, similar to a previous package, and clearly understood will need few, if any, storyboards. A rather large project, breaking new ground, and with many ideas floating around in a fuzzy fog will require many and more detailed storyboards.

Storyboards work through the human-computer interface. This HCI is often the troublesome portion of a new software development. A nice feature of the storyboarding method is the physical record produced by the storyboards themselves. They provide a ready reference for the design team as they begin the conversion of the "what" into the "how".

Some tips for storyboarding:

- Don't invest too much in a storyboard. The users need to be encouraged to make changes, and if the boards look too perfect, they will hesitate to suggest anything new.
- Make the storyboards easy to change.
- Don't make the storyboards too functional. A famous local professor tells the story of creating a "prototype" of a proposed software system. The customer was invited in to view this prototype and suggest changes. When the customer saw the prototype, he said, "That's fine, I'll take it home as it is". He didn't realize that there was no functionality behind the pretty screen displays.
- Whenever possible, make the storyboard interactive. Playing with an actual mouse, keyboard, and display will generate more feedback and elicit more new requirements.

In summary, the suggestion is to storyboard early, storyboard often, and storyboard on every project that has innovative content.

3.9 Stakeholders

Elicitation of a good set of requirements requires knowing all those who are involved with the project. These stakeholders can consist of state officials, any members of the development team assigned to the project, management, clerks, or any person who will use this system to do productive work.

A representative of all these groups should be included in the requirements gathering process. Leaving just one important stakeholder out of the discussion can cause a failure of the system late in the development cycle when it costs a lot to fix.

3.10 Use Cases

Often the hardest part in beginning to develop a set of software requirements is simply getting started. No other way is known than to simply take the first step. A technique that is used with UML is called Use Cases. These often get the user and developer off square one and into the writing process.

A Use Case is a very simple concept. It takes an actor playing a particular role and determines "what happens" when this actor performs some activity to be controlled by the developing software system. It could be as simple as a new user "logging on" to the system or a receivables clerk handling "back orders". In any case, the focus is very narrow and only the activities needed to do this particular Use Case are listed.

UML provides a diagramming technique to support Use Cases, but simple unambiguous or structured English on a yellow pad would suffice. A number of examples are provided at the end of this section.

To define a Use Case, do the following:

- Pick an actor from among the users (system manager, payables clerk, administrator, field technician, etc.).
- Pick a roll that this actor will be performing with the software system (logging on, entering a new vendor, accessing a payable account, adjusting an employees' pay rate, etc.).
- Then, thinking about that actor performing that role, what should happen?

3.11 "Mary Had A Little Lamb" Heuristic

Gause & Weinberg [Gau1989] propose a simple technique to elicit all the possibilities from a requirements statement. It is called the Mary Had A Little Lamb heuristic and is quite simple to operate. You just repeat the sentence you are analyzing putting emphasis on each work in sequence. Then you think about the other information that might be revealed by this emphasis.

- MARY had a little lamb.
- Mary HAD a little lamb.
- Mary had A little lamb.
- Mary had a LITTLE lamb.
- Mary had a little LAMB.

Then you continue with combinations until you have the entire sentence covered. Many of these words will add little or nothing to the understanding of the requirement. But, any insight gained is worth the small effort involved.

- MARY HAD a little lamb.
- Mary HAD A little lamb.
- Mary had A LITTLE lamb.
- …
- MARY HAD A LITTLE LAMB.

Try this on the requirements you are creating and watch what additional information is extracted.

4 Project Management

All software development life cycle methodologies suggest that there be some sort of plan for the project, but most leave the details to the interested reader. We have integrated the suggestions of the PMP [Hel2004] and other authors to make the requirements for the project plan more explicit.

A project plan has nine knowledge areas [Hel2004]:

- Integration Management,
- Scope Management,
- Time Management,
- Cost Management,
- Quality Management,
- Human Resource Management,
- Communications Management,
- Risk Management, and
- Procurement Management.

Most of these are obvious, but Project Integration Management can be best defined as a dynamic process that coordinates all of the other parts of the plan to insure consistency. We will not discuss this process explicitly, but it will be present in the ensuing discussions.

This report will not deal with Human Resources Management, Communications, or Project Procurement Management as they are out-of-scope.

This component is needed to meet the Software CMM requirement for Software Project Control at Level 2 and Integrated Software Management at Level 3 [Ahe2004].

4.1 Scope Management

Scope Management is concerned with defining and controlling the work of the project, so it incorporates product scope and project scope. There are five processes in this activity: Initiation, Scope Planning, Scope Definition, Scope Verification, and Scope Change Control.

For a software project, this reduces to translating the Requirements Document into a work plan and managing any ensuing changes. It begins in the planning phase with the development of a preliminary plan that assumes a project scope as detailed in a scope statement. The scope statement provides: a project justification, a product description, project deliverables, project objectives, and informal project requirements. The scope statement is refined in the System Analysis phase and is used to create the Work Breakdown Structure (WBS) which maps out the project deliverables and reduces each to a set of identifiable tasks.

The Requirements Document is superseded by the System Specification which may result in some updates to the WBS, and these changes will continue into the maintenance cycle. The Requirements Document, System Specification and WBS are living documents that will change continuously during the project and must at all times reflect the current understanding of the project team regarding product requirements and task assignments.

4.2 Time Management

Time Management has five processes: Activity Definition, Activity Sequencing, Activity Duration Estimation, Schedule Development, and Schedule Control. The Activity Definition begins with the WBS and Scope statement. In conjunction with historical data, constraints and assumptions about the project are used to reduce the WBS to a Task List of activities that are identifiable and assignable. There are no specific criteria for defining a task, but it should be small enough that the time estimate is accurate to within some acceptable margin of error. Early in the process, it might be reasonable to estimate development time as 3 months, assuming a possible error of 1 month, while later individual development tasks might be required to be estimated within 1 day. The Task List will be under constant change as the design and development continue. The Task List should contain fields for the following:

- The task identifier;
- The task description;
- The estimated duration: minimum, expected and maximum;
- Resources required;
- Estimated resource cost;
- Estimated total cost;
- Actual time required;
- Actual cost; and
- Notes from the project manager.

The second time management structure is the Precedence Diagram which shows the dependency relationship between the tasks. This document can show the project manager the appropriate assignments to make to avoid bottlenecks and stoppages. It is based on the time estimates in the task list and generates a set of paths through the project to completion. It is also common to perform a Critical Path analysis of the precedence diagram to determine which activities constitute the longest path through the task list, which is called the critical path.

As with the scope, the time management tools are dynamic and must be updated continuously to reflect new information and to collect data.

4.3 Cost Management

Cost Management has four processes: Resource Planning, Cost Estimating, Cost Budgeting, and Cost Control. Cost estimating produces data into the Task List and Precedence Diagram so there are no new documents, but successful project management depends on the collection and use of data for future estimates. Cost estimates are needed for the Benefit/Cost Analysis and for improving project cost estimates**.**

4.4 Quality Management

Quality Management has three processes: Quality Planning, Quality Assurance, and Quality Control. Quality in software can be objective or subjective. Objective measures are surveys of user satisfaction, lower operating costs, higher productivity, and lower defect and rework rates. It is important to collect as much information as possible, but the reality is that information outside of the development group is expensive to collect and is typically not done.

Most of these issues are outside of the scope of this document, but a defect tracking system is part of the process, as is user feedback on software, training, and documentation.

4.5 Risk Management

Risk Management has six processes: Risk Management Planning, Qualitative Risk Analysis, Quantitative Risk Analysis, Risk Response Planning, and Risk Monitoring and Control. Risk can be defined as potential costs in time or other resources that could be incurred. This can impact the SDLC in two ways: there could be risk associated with estimates of time or resources in the development process and there could be risks associated with factors outside of the SDLC, such as changing needs. External risks could be due to:

- Budgetary changes,
- Political issues,
- Legal issues, and
- Environmental issues.

The risks in the SDLC could be due to:

- Schedule slip,
- Scope slip,
- Technical issues, and
- Personnel issues.

Each of these presents some danger that the project will fail or will incur more costs than expected, possibly more than is tolerable. While these risks can only be partially controlled, they can be planned for. During the planning stage, a Risk Assessment will identify potential risks to the project, evaluate their potential impact and determine under what conditions a project might be halted or reviewed. At this stage, it might be necessary to halt a project if the risks are substantial.

During the System Analysis phase, a Risk Management Plan for the SDLC will be developed which will detail what risks are to be considered and how their impact should be evaluated with regard to the project. A Risk Assessment document prepared at each stage is used to evaluate the risk under the Risk Management Plan.

For example, during the System Design phase, the project team might determine that a particular task involves technology for which they lack experience. This is a higher than normal risk because the development time estimates, and possibly even the feasibility of the project, are in question. The team and project manager should evaluate this potential risk in determining how to proceed. One possible risk reduction strategy borrowed from Agile development is to have part of the team pursue this development until they are certain of the time and cost estimates and then use the new data.

The project management process is detailed on the web site.

5 Process Measurement

There are three reasons to use process measurement in a software project:

- To monitor project performance for management purposes,
- To collect data for performing estimates, and
- To identify and correct errors in estimating projects.

The subject of project estimation is not the subject of this analysis, but the collection of data for the three activities is similar and is a required part of the SDLC. An excellent reference for software measurement is found in [Mun2003].

The management of any process requires that the state of the process be measured and compared to some standard. Typically, this would be the collection of data that indicates progress towards goals which can be compared to plans. In order to have an accurate plan (or estimate), you need data. The chicken-or-egg analogy is easy to make, so you have to start with the best possible estimates and move towards ever-better data collection and estimates in an iterative manner.

The most important condition for success is the creation of a measurement culture that holds the collection of high quality and the proper use of it in high regard. This is a significant management problem that won't be addressed here other than to suggest that every part of an organization should be encouraged to view quantitative measurement as or performance as required. Often the best suggestions for measurement data come from those being measured. Every useful measurement that can be made without constraining productivity should be made and attention should be made to finding methods of collecting data that are neither intrusive nor draconian. Software development is a people process and people provide better data when they feel it is necessary and as simple as possible.

A metric consists of a measurement and a scale. There are three types of measurements:

- Direct, such as the number of lines of code;
- Indirect, such as the defect density, which is the number of defects divided by the code size; and
- Predictive, such as the expected time to completion based on the number of functions to be implemented.

And there are five types of scales:

- Nominal, which is simply a set of categories and has no measurement value. For example, Bob, Jane and Pat.
- Ordinal, a simple relativistic scheme. For example, high, medium and low.
- Interval, comparative by the size of the intervals. For example, three days longer.
- Ratio, a direct comparison, such as 2.5 times longer.
- Absolute, an exact measurement, such as 1000 lines of code or 50 defects.

For the most part, it isn't necessary to invest a lot of effort in these matters, but it is obvious that certain scales are more effective then others. An absolute scale measurement is better if you can get it, but if not, a ratio scale is better than an ordinal scale. Likewise, a direct metric is better than a predictive metric, if you can find one that will provide the necessary information. Nominal and ordinal scales are of less use because they do not yield numeric information that can easily be used for estimates.

5.1 Objectives

Before a Measurement Plan can be created for a project, the objectives of measurement must be determined. These are typically related to improving estimation, reducing project cost, and improving project management. For example:

- Reduce the cost of maintenance by 50%.
- Improve the timeline estimate by 20%.
- Reduce developer turnover by 25%.
- Reduce testing time by three weeks.
- Achieve a defect density of less than 0.5%.

A complete list of objectives should be prepared and then pruned to a reasonable set before attempting to create a Measurement Plan for a project. Trying to achieve too much can be frustrating for all concerned; also try to avoid conflicting objectives. For example, it would be difficult to attempt to reduce defects significantly while also reducing the time to release, although done incrementally, both may be achievable.

Once the objectives are determined, write 3-to-5 questions that you could ask to determine if you are reaching your goal. For example, if the objective is to reduce the length of the SDLC by 20%, you might ask:

- What percentage of time is spent in each phase of the SDLC?
- What percentage of the time is consumed by major categories of activities, such as design meetings, unit testing and so on?
- What is the rate of occurrence of development bottlenecks that reduce progress?
- How much time is spent in rework of code?

Now you know what you have to measure.

5.2 Metrics

From the above analysis, you can determine a set of metrics to be used to answer the questions and strive to reach the objectives. Of course, you can measure everything, but it is wise to be cautious. Analyzing data and putting it to work is time-consuming and too much data can be so noisy that you can't determine any meaning. You want to collect the minimal amount of data that tells you what you need to know because that minimizes the time spent providing and analyzing the data. So you want to choose the best metrics in the sense that they provide a good balance of the highest quality information and the easiest collection and analysis.

The metrics for software projects can be divided into two broad categories: by target group and by indicator. Target group can be identified as an individual team member, a team, or the development organization. Indicators are the part of the SDLC process being measured and might include progress, effort, cost, review results, trouble reports, requirements stability, size stability, resource utilization, and training. These can be varied depending on the organization and the project, but together they can form a matrix that makes it easy to insure the measurement objectives are being met.

When beginning to analyze data needs, keep in mind what you are trying to determine. Some examples of appropriate metrics for different types of target groups are shown in Table 1.0.

Table 1.0 – Examples of Appropriate Metrics for Target Groups

Target Group	Appropriate Metrics
Individual Developers	• Work effort distribution. • Estimated vs. actual task duration and effort. • Code covered by unit testing. • Number of defects found by unit testing. • Code and design complexity.
Project Teams	• Product size. • Work effort distribution. • Requirements status (number approved, implemented, and verified). • Percentage of test cases passed. • Estimated vs., actual duration between major milestones. • Estimated vs. actual staffing levels. • Number of defects found by integration and system testing. • Number of defects found by inspections. • Defect status. • Requirements stability. • Number of tasks planned and completed.
Development Organization	• Released defect levels. • Product development cycle time. • Schedule and effort estimating accuracy. • Reuse effectiveness. • Planned and actual cost.

Some examples of metrics types for different types of indicator categories are shown in Table 2.0.

Table 2.0 – Examples of Metrics Types for Indicator Categories.

Indicator Category	Objective	Metrics Types
Progress	Provides information on how well the project is performing with respect to its schedule.	Actual vs. planned task completions. Actual vs. planned durations.
Effort	Provides visibility into the contributions of staffing on project costs, schedule adherence, and product quality.	Actual vs. planned staffing profiles.
Cost	Provides tracking of actual costs against estimated costs and predicts future costs.	Actual vs. planned costs. Cost and schedule variances.
Review Results	Provides status of action items from life-cycle review.	Status of action item.
Trouble Reports	Provides insight into product and process quality and the effectiveness of the testing.	Status of trouble reports. Number of trouble reports opened, closed, etc. during reporting period.
Requirements Stability	Provides visibility into the magnitude and impact of requirements changes or *feature creep*.	Number of requirements changes/clarifications. Distribution of requirements over releases.
Size Stability	Provides insight into the completeness and stability of the requirements and into the ability of the staff to complete the project within the current budget and schedule.	Size growth. Distribution of size over releases.

Computer Resource Utilization	Provides information on how well the project is meeting its computer resource utilization goals/requirements.	Actual vs. planned profiles of computer resource utilization.
Training	Provides information on the training program and staff skills.	Actual vs. planned number of personnel attending classes.
Software	Measures the quality and reliability of the software product.	Code size. Code complexity.

5.3 Collection and Presentation

The collection of measurements must be carefully planned to be successful. The method of collection must be determined, and the procedures for collecting and storing the data must be defined and communicated to the appropriate personnel. For each metric, the following information should be determined:

- The exact data to be collected. For example, the hours spent working on the removal of defects reported by the defect management system.
- The schedule for collecting the data: daily, weekly, on demand, etc.
- The method to be used for collecting the data: paper form, email, web site form, and/or interview. Insure that the methods are operational and that procedures for handling the communication are in place.
- The person or persons responsible for collecting the data.
- How and where the data is to be stored pending analysis and in what format.
- What analysis is to be performed on the data? For example, the metric may be the result of averaging a data set and dividing by another collected data item, or a regression analysis may be used.
- Describe the presentation method. For example, a graph, table, written analysis or some combination. Be precise as to the exact content of the reporting schema and what comparisons are to be made between metrics.
- Describe the reporting method. How often reports are to be produced, who is responsible for reporting the data and who is to receive the reports.

5.4 Privacy

It is vital that privacy of data be properly observed. Individual data should not be viewable by other individuals, and team data, to as great an extent as possible, should not be shared globally. This is not to say that the data can't be used as an incentive, but handling data properly is a significant management problem. Remember that to get good data, people have to feel that it will be properly used.

5.5 Possible Metrics

There are innumerable metrics that could be used, but the following are examples of suggested data points that are generally considered valuable. The list is organized by metric type.

1. Software size
    a. Source lines of code
    b. Code coverage
    c. Rate of source code change (code churn)
    d. Rate of growth of source code
    e. Function points
    f. Bang (see DeMarco [Dem1999])
2. Software complexity
    a. Cyclomatic complexity
    b. Number of classes and interfaces
    c. Knots
    d. Information flow
    e. Halstead's Program Vocabulary
    f. Halstead's Program Length
    g. Halstead's Program Volume
    h. Cohesion
    i. Coupling
3. Software quality
    a. Defect rate (defects/lines of code)
    b. Defect classification distribution
    c. Pass/fail rates for integration
    d. Failure modes in acceptance testing
    e. Number or rate of design changes
    f. Defect rates by stage
    g. Cost of defect remediation
    h. Defect rate during maintenance
    i. Complexity measures
    j. Module size
    k. Robert Cecil Martin's metrics
4. Design metrics
    a. Number of requirement change orders
    b. Number of interface definition change orders
    c. Inputs per Use Case
5. Process cost
    a. Per phase cost
    b. Staffing per phase
    c. Computer resources utilized
    d. Space resources utilized
    e. Amount of rework required
    f. Staff function distribution

    g. Staff rate of growth
    h. Cost of quality assurance
6. Other
    a. Page of documentation

6 Software Design

This component of the project comes exclusively from the USDP and is concerned with developing an Architectural Model and Design Model based primarily on the Requirements Document. The researchers propose using UML for these processes as is used in the USDP and documented in [Fow2004], [Mak2005], and [Rum2005]. The methods described here meet the Software CMM requirements for Software Product Engineering at Level 3.

The Unified Software Development Process [Jac1999] describes the process as the development of a series of models. The process models of interest to this study and their relationship to the USDP and MDT Phases are shown in Table 3.0.

Table 3.0 – Unified Software Development Process Models

USDP Phase	Model	MDT Phase
Business Planning Phase	Business Use Case Model	Planning Phase
Requirements Phase	Requirements Use Case Model	System Analysis Phase
Analysis Phase	Analysis Model	System Design Phase
Design Phase	Design Model and Deployment Model	Technical Design Phase
Implementation Phase	Implementation Model	Development Phase
Test Phase	Test Model	Implementation Phase

While the MDT SDLC varies slightly in terminology and organization, the models are exactly what the MDT requires. The properties of the models are:

- A Use Case Model is a representation of system requirements as Use Cases and Actors.
- An Actor is an entity that participates in the system under consideration. An Actor could be a user or controller of the system, another software system or a device. Anything external to the system that plays a role in interacting with the system.
- A Use Case is a method for capturing the functional requirements of the system by specifying the typical interactions between the Actors and the system.

Appendix A contains a full example of a Use Case Model.

6.1 Planning Phase

A Business Use Case model (see Table 3.0) identifies the context in which the software system operates in the business environment. It depicts what is outside of the business system, what is inside of the business system, and how they are related. In this sense, the Business Use Case is different than a Requirements Use Case which attempts to depict how the software interacts with the outside world. The Business Use Case Model is a set of all known Business Use Cases for the system and it represents a very high-level

rendition of the system requirements. The requirements development process is covered in detail in Section 3.

6.2 System Analysis Phase

The Requirements Document (see Table 3.0) developed in this phase is used to initiate the system analysis and provides the basis for the Analysis Model. The requirements provide all the information necessary to design the system (assuming they have been done correctly and don't change), but they must be written in the language of the customer and that is not particularly useful for developing software. In the System Design phase, the Requirements Document is converted into a System Specification and then into an Analysis Model.

The primary tool for describing requirements in the USDP is the Use Case**,** but we also propose the use of textual requirements that can be easily understood by all people involved in creating the Requirements Document and to insure that the diagrams are consistent and complete. It is easy to miss something or to assume that a diagram is describing something that isn't. This is extra work, but will avoid problems later on in the process.

The development of requirements is covered in detail in the Requirements Development section of this document.

6.3 System Design Phase

This is a design phase in the MDT SDLC (see Table 3.0) but it corresponds to the System Analysis phase of the USDP, so the terminology can be confusing. This is high-level design and so falls into the category of analysis and not software design. The Analysis Model is to the Use-Case Model what the System Specification is to the Requirements Document; a more software-developer oriented version of the system. The USDP considers the Analysis Model to be sufficient, but experience indicates a System Specification that details the required behavior in some English form is a valuable check on the UML description of the system and requires the designers to match the requirements twice. While this does require an investment of time, mistakes at this stage can be extremely costly later in the process. In general, the System Specification should contain exactly what the Requirements Document contains, and if it doesn't, the Requirements Change process should be used to make necessary changes. The difference is that the specifications are written in the language of the software programmer and are much more precise.

The Analysis Model seeks to avoid doing any design and to focus on restructuring the Use Case requirements so that they are representative of the software development concerns of the team. Use Cases are expanded into Use Case realizations that expand the requirements by:

- Focusing on functional requirements and responsibilities,
- Ignoring user interfaces,
- Avoiding any discussion of implementation specifics,

- Concerning itself with conceptual relationships, and
- Dealing only with a limited set of interactions between system components.

Another way to view the Analysis Model is that it could never be used to directly write software and if it could be, it has too much detail. The primary activity in System Design is converting Requirements Use Cases into Use Case Analysis Realizations. This is accomplished by using a combination of the following constructs:

- Analysis Class Diagram,
- Class Diagram,
- Communication Diagram,
- Flow of Events,
- Special Requirements,
- Analysis Package,
- Service Package, and
- Architecture Description.

Analysis Class Diagram
Class diagrams are used throughout the analysis and design phases, but at this point, they are focused on the high-level functionality. There are three types of classes that can be found in the Analysis Model: boundary classes, entity classes and control classes. Boundary classes are used to model the interaction between a system and its actors.

Entity classes are used to model information that is long-lived and possibly persistent.

Control classes represent coordination, sequencing, transactions and control of other objects and typically represent the control mechanisms of an individual Use Case.

Communication Diagram
As discussed earlier, these diagrams show the communication links between objects in the design. They are important here in understanding the data that passes between the classes and other design components. Communication diagrams provide detail concerning the exchange of information (at a very high level) between the components of a system and the system and actors. Communication Diagrams used to be called Collaboration Diagrams in UML 1.x.

Flow of Events
These are textual representations of the Communication Diagrams or other forms of Interaction Diagrams, which are often difficult to understand completely because they provide no temporal information.

Special Requirements
These are textual descriptions of any non-functional requirements in the Use Case description of the system or any requirements discovered during the analysis.

Analysis Package
These are subsets of the Analysis Model that group parts of the system based on common functionality; they help organize a large model into manageable pieces. Different Analysis Packages should not be based on any solution domain or on non-functional criteria.

Service Package
A Service is a collection of functionally related actions that are employed in more than one Use Case. In other words, it is a unit of functionality that is reusable. This is obviously important in obtaining the benefits of reusability in your software design.

Architecture Description
An Architecture Description is a combination of diagrams and textual information that provides a high-level view of the Analysis Model, consisting primarily of Analysis Packages and their dependencies, key classes and critical Use Cases.

In addition to the USDP, we add the need to develop user interface storyboards that provide a high-level description of the components of the interface users will see and the sequence of operations users will expect to see. These develop naturally from the Use Cases and identifying the characteristics of the interface early can provide an opportunity to get feedback from users before doing significant design work. An example of a storyboard is provided in the Requirements Section.

We also add the plan for acceptance testing in this phase, which requires the development of a Test Model, composed of multiple instances of Test Cases, Test Components, and Test Procedures. A Test Case specifies how a Use Case is to be tested. This is typically done in some textual form and provides the data inputs and the expected results that are to be tested, as well as the conditions on the test.

A Test Procedure specifies how to perform one or more Test Cases. Note, the Test Case specifies what is to be tested and the Test Procedure specifies how the test is to be conducted.

A Test Component is a structure that automates one or more Test Procedures, typically by scripting or with an automated testing tool.

6.4 Technical Design Phase
This phase elaborates the Analysis Model (see Table 3.0) and engages in a series of construction iterations that result in a Design Model**,** which is a stable architecture and blueprint for development. The Design Model is much closer to the physical manifestation of the software and has to be more specific and formal than the Analysis

Model. Once created, it has to be maintained throughout the development cycle to insure the software continues to represent the underlying design and the design itself is consistent with good software standards.

In the USDP, the purpose of this phase is to:

- Decompose the Analysis Model into manageable pieces that can be assigned to the development team (possibly concurrently) while insuring that the requirements are met.
- Capture major interfaces between subsystems.
- Identify issues regarding non-functional requirements and constraints.
- Create a point of departure for implementation.

The UML constructs used in the Design Model are:

- Design Class,
- Use Case Realization-Design,
- Class Diagram,
- Sequence Diagram,
- Flow of Events,
- Implementation Requirements,
- Design Subsystem,
- Service Subsystem,
- Interface,
- Architecture Description, and
- Deployment Model.

Design Class
A Design Class differs from an Analysis Class in that the description language is similar to that of the programming language to be used. It need not be syntactically correct, but it should be a near representation of major syntactical constructs such as function calls, parameters, operations, and data types. It does not, however, deal with implementation issues such as specific coding methods or graphical user interface components.

Use Case Realization-Design
This is similar to an Analysis Use Case Realization in that it provides a path between the original Use Case and its requirements and the Design Model.

Class Diagram
This construction shows the high-level relationship between a Use Case and the derived Design Classes. This is important in insuring system requirements are being met and to avoid confusion over the handling of requirements.

Interface
An Interface is a representation of the operations provided by the associated Design Class or subsystem. They are a way to separate the internal workings of a Class from the external view of what functions the Class offers to other Classes or Subsystems. They might be viewed as a description of Application Program Interface.

Sequence Diagrams
These constructs show how the flow of data and locus of control changes as a Use Case is performed. These are the counterparts of the Collaboration Diagrams used in the System Design phase.

Flow of Events
These are textual representations of the Sequence Diagrams, which do not provide information about the interaction between Sequence Diagrams representing different Use Cases.

Implementation Requirements
These are textual representations of implementation requirements that cannot otherwise be represented. They may flow down from other phases or they may evolve in the Technical Design phase.

Design Subsystem
These are artifacts of decisions about managing the design or subsequent phases. Design Classes, Use Case Realizations, Interfaces, and other Subsystems can be grouped due to similarity in function, development team assignments or any other criteria that makes sense. A subsystem should be cohesive and different subsystems should be coupled only loosely.

Service Subsystem
As in the Analysis Model, a Service is a collection of functionality that has common usage in more than one location in the Design Model.

Architecture Description
This is a high-level description of the Design Model showing the decomposition into Subsystems, Classes, Interfaces and Use Case Realizations.

Deployment Model
This model describes the physical distribution of the system across computational resources.

Unit Test Plan
In this phase, we suggest the development of the Unit Test Plan**,** which specifies the tests that each unit of the software has to meet. These could be tests on individual methods or classes, or on conglomerates that provide some function. During the design phase, as each class or subsystem is created, its functionality should be carefully spelled out, based

on the System Specification, and tests should be written that when successfully completed, will verify that the software works as it should. There are two types of tests:

- Black-box tests where the unit's observable external behavior is validated.
- White-box tests which validate the unit's internal implementation.

The former are usually done by writing a program that exercises the unit and tests that all the methods or functions work as they should. The latter are usually done via code walkthroughs and/or outputting intermediate results when the unit runs.

Defect Tracking System

It is absolutely essential that a software project have a defect tracking system to insure defects are tracked and fixed and historical data on defects is accessible. The tracking system could be as simple as a bulletin board where defects are posted and progress tracked, but the need for historical data indicates that an electronic form is preferable. There are a number of commercial or open source products for defect tracking, but an electronic bulletin board or web site, or even a commonly accessible spreadsheet can be used. The system should keep the following data:

- A defect identifier,
- The date submitted and the date of all state changes,
- The defect description,
- Who submitted the defect,
- Who has been assigned the defect,
- Supporting documentation for the defect,
- The location of the defect in the code when possible,
- The expected result of fixing the defect,
- What has been done to fix the defect (workarounds, updates, etc.), and
- The current state of the defect (open, fixed-not-verified, closed).

The supporting documentation could include attachments including screen shots, core dumps or other evidence.

As the defects are fixed, the type and number of defects can be understood, helping to improve the development process, and the defect list can help in understanding the current capabilities of the software.

A defect tracking system is more complex than a simple form so it is discussed in a separate section.

Deployment Management Plan

During this phase the development team must determine how the completed software will be deployed and managed by the hosting team (internal or external). Some of the decisions have already been made. The Deployment Plan dictates if the software runs as a stand-alone application, web application, or in some other manner. CM plans are not

included in USDP or in the Software CMM, but they are part of the IEEE 828-1990, NASA-Sfw-DID-04 and DoD-STD-2167A [SEI2004].

Configuration Management Plan
A Configuration Management Plan (CMP) provides a basis for identifying changing software versions. A CMP should include:

- An identification scheme for versions,
- Methods for tracking, auditing and controlling changes,
- An organization for storing software,
- A methodology for testing software, and
- A methodology for maintaining stable releases and baselining at milestones.

This plan helps control the software development process and the following maintenance process. Without an adequate CMP, it is nearly impossible to manage a constantly changing software cycle.

The identification scheme should represent a reasonable representation of the type and frequency of updates that are expected, but even in the development phase, versioning should be used. Some examples are the simple *version.release* form (5.1), while software that is subject to more frequent updates might need *version.subversion.release* (5.1.4). Some organizations prefer to use letters (5.A.d) or even names (WOMBAT.4), but the important issue is whether it has sufficient resolution to meet the needs for clearly defining software changes internally.

The storage and control of the software under development and during maintenance should be done by a version control system of some kind. This is a decision that is very much a matter of institutional choice, but it should provide the following:

- Sufficient file space for the project and frequent backups to another media,
- The ability to set milestone versions and check them out at any point in time,
- The ability to rollback in the case of a significant problem, and
- Versioning that can match the versioning system chosen for the software.

The remaining need is for a testing methodology. The Acceptance Test Plan, System Test Plan, and Unit Test Plans are concerned with the types of tests to be run and the desired result, but the CM plan needs to specify the exact methodology that is to be used for testing. The possibilities are endless, but are typically one of these types:

- Continuous testing,
- Testing only when the software reaches a particular milestone,
- Testing only when the software has reached the final state and includes all functionality, or
- Some combination of these, such as continuous unit testing, system tests are all milestones, and acceptance testing at the end.

The nature of unit testing is that it should take place within the development cycle and should be done at short intervals, possibly by the programmers themselves. Acceptance tests are typically done only at the end by the customers, although dry runs are often concocted to insure compliance. System tests can occur at any time, but it is generally agreed the more frequent they are the better.

Quality Assurance
One issue that is difficult to settle in some organizations is if testing and quality assurance should be done by the developers or by a group of people not directly in development. It is our opinion and the opinion of most in industry ([Spo2001], [McC1993], and [McC1996]), that a separate quality assurance or testing group is both cost-effective and more effective in the quality sense.

The next question is how quality assurance (QA) should be integrated into the SDLC. It is generally viewed as a process that begins when development ends, but that is not particularly effective and should be avoided. Doing continuous QA while development continues identifies problems before they become highly dependent parts of the code, it allows better interaction with clients if requirements change and it tests the entire system, including configuration and implementation well before the client sees it.

6.5 Development Phase

In this phase (see Table 3.0), the Design Model is converted into running and tested software. The USDP suggests that the following functions be incorporated:

- Plan the system integration process, which is a series of iterations of development. The software can be developed top-down, bottom-up or by stove-piping as in Agile development methods.
- Implement the classes, subsystems and interfaces from the Design Model.
- Unit test the components.

The UML constructs used in this phase are:

- Component,
- Interfaces,
- Architectural Description, and
- Integration Build Plan.

Component
This structure is a physical packaging of design classes, interfaces, and subsystems representing the manifestation of the design phase. A Component could be an executable program, a file containing data or code, a library, a database table, or a document, as well as anything else that is needed to implement the design.

Interfaces
In this phase, an Interface is virtually the same as it is in the Analysis Phase.

Implementation Subsystem
This is a collection of implementation artifacts used for organizational convenience. An Implementation Subsystem should have a physical, programming language manifestation, such as a Java package, Visual Basic project, or Component View Package in Rational Rose.

Architectural Description
This is a high-level view of the Implementation Model showing the significant artifacts, including Components, Implementation Subsystems, Interfaces, and dependencies.

Integration Build Plan
While this is planning, it is an integral part of the development cycle and is necessary to insure that appropriate methods are used in development. An IBP decomposes the development cycle into a series of milestones or builds which are executable versions of the system or some part of the system. Each build should be testable and there should be a test plan for each. This idea is that incremental implementation is easier to manage and test.

The Integration Build Plan describes the functionality to be included in each build, which components of the implementation are affected and the test plan. This is closely related to the entire project planning process but rightly belongs here.

How you proceed from the Design phase to the Development phase is dependent on a number of factors but it reduces to the following issues:

- Where do you start?
- How do you proceed?
- When do you stop?

Starting may happen before the end of the design phases, primarily based on the issue of risk. If a particular design component appears to be risky in the sense that it involves unknown technology or there is some doubt about how the implementation might affect the design, it should be started and pushed to a point where the risk is either mitigated or the project is reviewed for feasibility. Whenever it occurs, it is best to remove any questions about the development and implementation as soon as possible.

If there are no such issues, the design should be organized around code reusability; parts of the design that are identified as reusable should be coded first so that the remaining code can be built on these functions. Beyond that, the code can be developed layer-by-layer, or by stove-piping. In layer-by-layer design, the code is developed from lower layers to higher layers so that testing can be done as the project progresses. In some cases, it might be easier to “drill down” through the code, completing entire portions of the project from the highest to the lowest layers of functionality so that those portions can

be thoroughly tested and possibly even released for use. In general, there are not industry standards that dictate the "best practices" and these decisions need to be made by the project team given the environment and structure of the design. However, these decisions must be made and not allowed to happen at the whim of the developers themselves.

Stopping seems obvious, but projects of any size tend to continue *ad infinitum* unless the deadline is reached. One of the advantages of continuous or frequent testing is that as the functionality nears what was promised, it becomes apparent the project should either diminish in intensity or new functionality should be added. Most projects move into the maintenance phase at this point, so development doesn't stop, but it moves into fixing bugs and adding functionality, usually at a different pace. The project manager should evaluate the project at every test point and determine when the project has reached fruition.

6.6 Implementation Phase

This phase (see Table 3.0) makes the software available, possibly in stages such as beta, early release, and final and performs the acceptance test. In the USDP, this phase is called Test and focuses only on the testing. For the MDT, this phase focuses on training and acceptance testing and the deliverables are:

- Training surveys,
- User response surveys,
- Completed acceptance test,
- Updated and tested software, and
- Updated user support plan.

This component is not specified by any standard process, but represents a collection of activities that are vital to long-term success and quality control. The Software CMM specifies requirements for an Organization Training Program at Level 3 and a Software Configuration Management at Level 2.

After the software is installed and tested to insure that it is operational, although not necessarily defect-free, the training can take place and the user surveys can be collected. The development team may or may not be involved in training, but this is typically a busy time for developers as they endure one more round of intense defect handling. The Acceptance Test may reveal problems that need to be resolved and this phase may extend over several months, depending on the complexity of the testing and the quality of the software. At some point, the software is accepted and it moves into the next phase.

When working with users during training and during the testing, there are always issues that arise concerning problems that are not critical to passing the test, but are significant for usability and user satisfaction. These should be reported through the defect tracking system to insure that they are recorded and considered for future changes.

A plan for supporting users can be developed which details how users with questions or problems should be handled. Defects go into the defect tracking system, but they need an avenue for submitting them, either through a web site, email, or telephone call. This needs to be spelled out. If they have questions, you may have a support team that needs to be trained on the product. Or you might assign a member of the development or maintenance teams to handle questions. You might also restrict access to a single user so that all questions have to be vetted by the user's organization to reduce support traffic. This is all dependent on organizational structures and resources, but it is very important that every question or defect be tracked and the resources used to resolve it measured. As discussed in the Project Planning section, the User Support Plan should include internal accounting of the cost.

6.7 Conclusion Phase

For MDT, this phase is about reaching the state where the development effort is over and ready to move into the maintenance phase, and the project is ready for review. This phase doesn't end until the software is no longer being updated or used. The deliverables are:

- Customer approval,
- Software in maintenance cycle, and
- Analysis of project performance.

This phase should not be reached until the Acceptance Test is complete and the training has been finished, so the first item should be completed. The software moving into the maintenance cycle means that a team of support personnel will take over the day-to-day care of the product and the defect tracking system. The analysis of the project should be primarily a report from the project manager indicating how well the team did according to its own estimates and any standards of performance used, and what procedural changes should be made to the SDLC in the future. This is discussed in more detail in the Project Management section.

6.8 Risk and Agile Development

The USDP encompasses a wide range of methodologies based on the concept of iterative and incremental development. The approach to be used depends on the organizational preference, the type of project being considered (rework, a large new development, a small project, etc.), and the decisions of the development team. Another set of methodologies are the Agile Development or Rapid Deployment methods, which take a different approach. It is the researchers' opinion that these methods do not currently represent industry best practices, and while they have shown promise, there have also been some failures and should be used with caution.

What the Agile methods provide are techniques that address development in situations where time is of the essence, but they require the order of development to be based on the evaluation of priorities and risk, and that can be valuable in any project.

7 User Interface Development

Although a part of software development, user interfaces (UI) are designed and implemented in a manner which is unique. As the part of the software that users see, the user interface conveys the solution to the Requirements document and is largely responsible for the perceived usability. User interfaces can be text-based, graphical, web-based, or possibly non-existent, but they must be considered carefully because they can become a significant part of the development and maintenance cost. They must also be carefully vetted by the users during the design and development process because there is so much subjectivity in the simple question, "Do you like this interface?"

The steps in developing the UI's for a system are:

1) Use Case Model,
2) User interface storyboards,
3) User interface mock-ups, and
4) Implemented user interfaces.

The UI development process starts during the System Analysis phase in the Use Case Model. Each Use Case represents the relationship between Actors and parts of the system. If the Actor is a person, you have a UI; if it is another system, you have an API or protocol. If it is a UI, it should be specifically determined what type of UI is desired: text, graphical, or anything else that is possible.

During the process of testing the analysis with the client, each potential UI should be discussed and there must be an agreement on the requirements for the UI's. For example, should they have menus, do they have a UI model in mind, should they have keyboard-only input to speed typing and so on. Of course, much of this discussion should happen during the Planning phase since the type of user interface impacts cost.

A user interface storyboard is a device for describing the interaction between the system and the user and it typically derives from Interaction and Collaboration Diagrams developed during System Analysis, as well as from the Use Cases.

During the Technical Design phase, the storyboard is converted into a user interface mock-up, which is an implementation of the UI in a form that is similar (possibly identical) to the final implementation. The purpose is to demonstrate the functionality and aesthetic value of the UI for customer approval prior to moving to final development. Agile development methods would build the actual interfaces with minimal backing capability, but this can be accomplished by using a rapid prototype development system, such as Visual Basic.

The mock-ups should be approved by the customer before proceeding. The storyboards show the messages and the sequence of collaboration, but they do not provide the users with a sense of how the interface "looks", but the mock-ups do, so the client has now approved all parts of the UI. Proceed with implementation of the UI.

8 Software Development

This component of the project is described in broad terms by the USDP and is concerned with converting the Design Architecture into running software. This phase is difficult to describe in detail because it depends significantly on the languages used, the target software, and hardware platforms and the underlying goals, but there are standard methods for managing this process that can be proposed. The methods described here meet the Software CMM requirements for Peer Reviews at Level 3 and Software Quality Assurance at Level 2.

The material here represents a set of methods considered to be industry best practices, but the use of them cannot be scheduled or determined precisely except by the development team as part of the planning and development process. They are provided here as methods to be considered in making choices about how the development process is to proceed.

8.1 Code Walkthroughs

A method for improving code quality is to use walkthroughs, where one or more programmers look at the code of another programmer, looking for violations of coding standards and problems in the logic. It requires the programmer to explain the code on a line-by-line basis and provides more eyes to look for potential problems. Often, the biggest improvement is in the overall improvement in coding style and design when programmers share ideas.

8.2 Programming Pairs

Another type of programming uses two programmers working together on a single task. They design together, split the coding and then test each other's work, including walkthroughs. There are potential personnel conflict problems with this method, but when it works, it has very good results.

9 Documentation

During this phase the development team develops the product and user documentation according to the specifications established earlier in the project process. The documentation process starts in the System Design phase with a Documentation Plan which describes the things which are to be done with regard to documentation:

- A list of deliverables;
- An identification scheme for the deliverables;
- Procedures for verifying the contents, publishing and transferring of each deliverable;
- For each document, a table of contents; and
- A schedule of milestones for the deliverables and work assignments.

10 Training Plan and Training

A training plan provides a framework for training users and support personnel in the operation and support for the product as specified in Software CMI GP 2.5 [Ahe2004]. A training plan for users should include the objectives, strategy, and curriculum to be used when training or developing training materials. It should also cover a variety of mundane details such as scheduling, reserving resources, publishing of training materials, and anything else necessary. Since training involves customer staff, coordination is important and they must approve of the plan.

A training plan is naturally the result of the skills of the users and the complexity (and quality) of the software. The plan should address:

- The objectives of the training,
- The approach to be used to develop the curriculum,
- A needs and skills analysis of the users,
- The training methodology,
- The training database,
- Methods for testing training effectiveness during development,
- Course resources (administration, space, equipment, materials),
- Training schedules, and
- Continuing training.

11 References

[Ahe2004] Ahern, D., et al, *CMMI Distilled*, Addison-Wesley, 2004.

[DeM1999] DeMarco, T., Lister, T., *Peopleware*, Dorsett House Publishing, 1999.

[Fow2004] Fowler, M., *UML Distilled, Third Edition, Addison-Wesley Pearson,* 2004.

[Gau1989] Gause, D., Weinberg, M., *Exploring Requirements: Quality Before Design, Dorsett House Publishing,* 1989.

[Gla2003] Glass, R., *Facts and Fallacies of Software Engineering,* Addison-Wesley Pearson, 2003.

[Hel2004] Heldman, *Project Management Professional Study Guide, 2nd Edition*, Sybex, 2004.

[Jac1999] Jacobsen, I., et al, *The Unified Software Development Process*, Addison-Wesley, 1999.

[Lef2003] Leffingwell, D, Widrig, D., M*anaging Software Requirements: A Use Case Approach, Second Edition,* Addison-Wesley, 2003.

[Kru2000] Krug, S., *Don't Make Me Think*, New Riders Press, 2000.

[Mak2005] Maksimchuk, R., Naiburg, E., *UML for Mere Mortals*, Addison-Wesley Pearson, 2005.

[Mar2005] Marasco, J., *The Software Development Edge*, Addison-Wesley Pearson, 2005.

[McC1996] McConnell, S., *Rapid Development*, Microsoft Press, 1996.

[McC1993] McConnell, S., *Code Complete*, Microsoft Press, 1996.

[Mun2003] Munson, J., *Software Engineering Measurement*, Auerbach Press, 2003.

[Pre2001] Pressman, R., *Software Engineering,* McGraw, 2001.

[Rum2005] Rumbaugh, J., et al, *The Unified Modeling Language Reference Manual,* Addison-Wesley, 2005.

[SEI2004] Software Engineering Institute, *CM Plans: The Beginning of Your CM Solution*, http://www.sei.cmu.edu/legacy/scm/abstracts/abscm_plans.html.

[Spo2001] Spolsky, J., *User Interface Design for Programmers*, Apress, 2001.

[Spo2004] Spolsky, J., *Joel on Software*, Apress, 2004.

Appendix A - UML Example

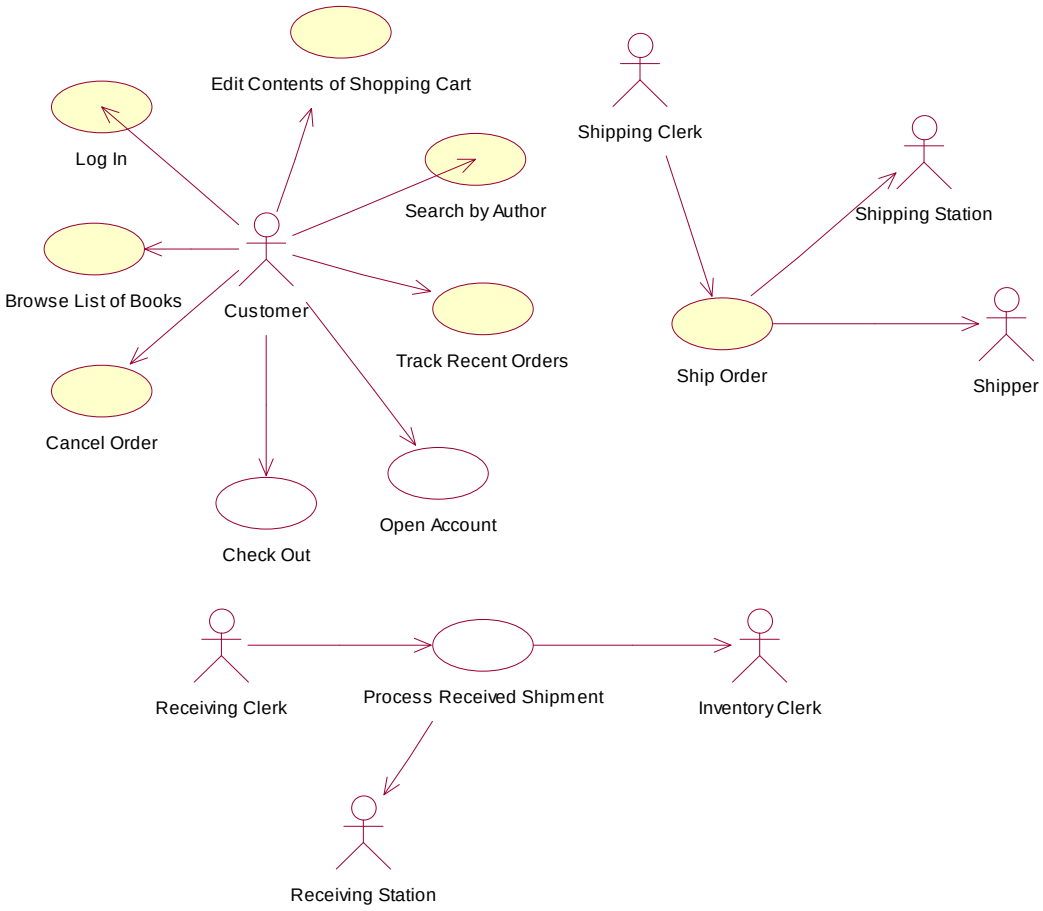
This is an example of a UML development process for a bookstore. It is neither too simple nor too complex and illustrates the modeling approach quite well. The UML Model report is included here. Also the "UML Model.doc" and four diagrams are included as .pdf files in the Reports directory of the supplemental software. For more details about this case study, see [Ros1999] and [Ros2001].

UML Model Report

UML Model

Use Case MODEL Report
Use Case Model
Package Documentation: *NONE*

Use Case Diagram - Main



Actor - Customer
Documentation: *NONE*

Actor - Shipping Clerk
Documentation: *NONE*

Actor - Shipper
Documentation: *NONE*

Actor - Receiving Clerk
Documentation: *NONE*

Actor - Inventory Clerk
Documentation: *NONE*

Actor - Shipping Station
Documentation: *NONE*

Actor - Receiving Station
Documentation: *NONE*

Use Case - Add Item to Shopping Cart
Documentation: *NONE*

List of Associations
Search Results Page Communicates with Add Item to Shopping Cart.

Use Case - Browse List of Books
Documentation:
Basic Course

The Customer clicks on a Category on the Browse Books Page. The system displays the subcategories within that Category. This process continues until there are no more subcategories, at which point the system displays the Books in the lowest subcategory. The Customer clicks on the thumbnail for a Book. The system invokes the Display Book Details use case.

Alternate Course

If the system does not find any Books contained within a given Category, it displays a message to that effect and prompts the Customer to select a different Category.

List of Associations
Customer Communicates with Browse List of Books.

Use Case - Cancel Order
Documentation:
Basic Course

The system ensures that the Order is cancellable (in other words, that its status isn't "shipping" or "shipped"). Then the system displays the relevant information for the Order on the Cancel Order Page, including its contents and the shipping address. The Customer presses the Confirm

Cancel button. The system marks the Order status as "deleted" and then invokes the Return Items to Inventory use case.

Alternate Course

If the status of the Order is "shipping" or "shipped," the system displays a message indicating that it's too late for the Customer to cancel the order.

List of Associations

Customer Communicates with Cancel Order.

Use Case - Check Out

Documentation:

Basic Course

The system creates a Candidate Order object that contains the contents of the Customer's Shopping Cart. Then the system retrieves the Shipping Addresses associated with the Customer's Account, and displays these addresses on the Shipping Address Page. The Customer selects an address, and then presses the Use This Address button. The system associates the given Shipping Address with the Candidate Order. Then the system displays the available Shipping Methods on the Shipping Method Page.

The Customer selects a shipping method, and then presses the Use This Shipping Method button. The system associates the given Shipping Method with the Candidate Order. Then the system displays the contents of the Billing Info objects associated with the Customer's Account, on the Billing Information Page. The Customer selects a billing method and presses the Use This Billing Information button. The system associates the given Billing Info object with the Candidate Order. Then the system displays the Confirm Order Page.

The Customer presses the Confirm Order button. The system converts the Candidate Order to an Order and destroys the Shopping Cart. Then the system returns control to the use case from which this use case received control.

Alternate Courses

If the Customer has not already logged in, the system invokes the Log In use case.

If the system does not find any Shipping Addresses, it invokes the Create Shipping Address use case.

If the system does not find any Billing Info objects, it invokes the Define Billing Information use case.

If the Customer presses the Cancel Order button at any time, the system destroys the Candidate Order and returns control to the use case from which this use case received control.

List of Associations

Customer Communicates with Check Out.
Shopping Cart Page Communicates with Check Out.

Use Case - Edit Contents of Shopping Cart

Documentation:

Basic Course

On the Shopping Cart Page, the Customer modifies the quantity of an Item in the Shopping Cart and then presses the Update button. The system stores the new quantity and then computes and displays the new cost for that Item. The Customer presses the Continue Shopping button. The system returns control to the use case from which it received control.

Alternate Courses

If the Customer changes the quantity of the Item to 0, the system deletes that Item from the Shopping Cart.

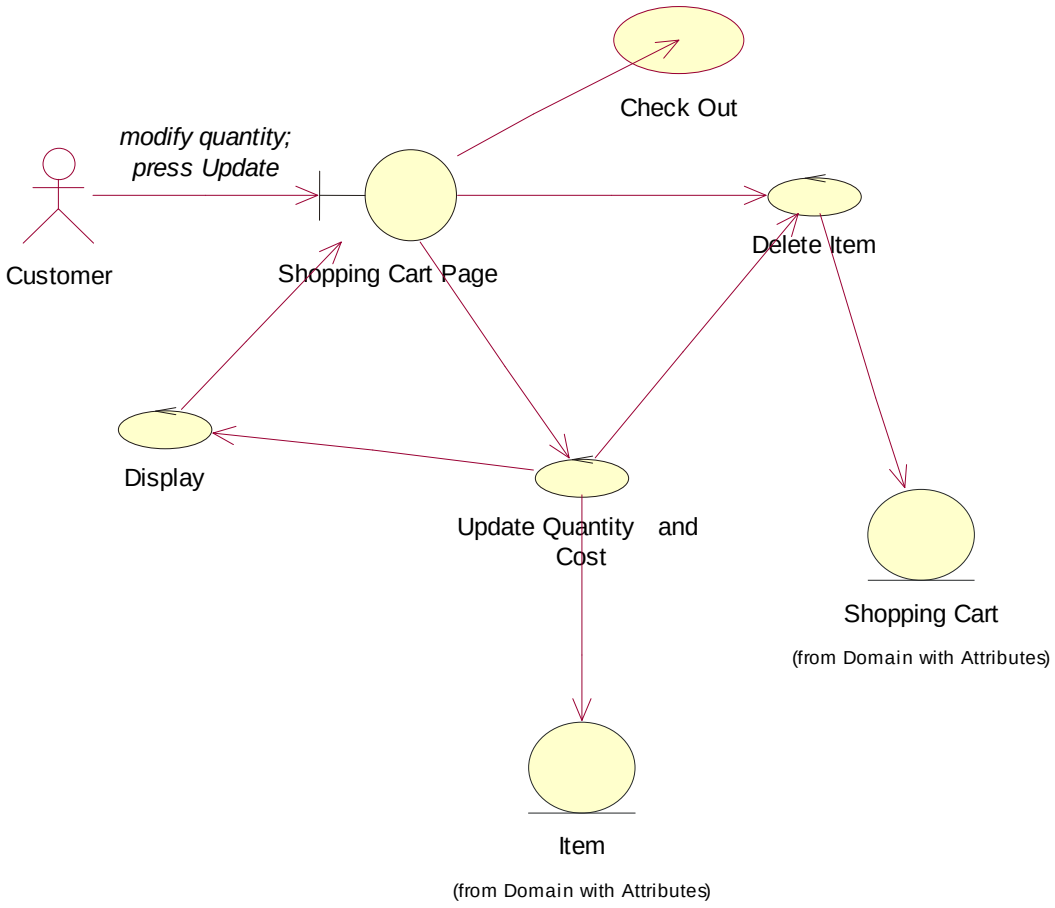
If the Customer presses the Delete button instead of the Update button, the system deletes that Item from the Shopping Cart.

If the Customer presses the Check Out button instead of the Continue Shopping button, the system passes control to the Check Out use case.

List of Associations

Customer Communicates with Edit Contents of Shopping Cart.

Class Diagram - Edit Contents of Shopping Cart Robustness



Interaction Diagram - Edit Contents of Shopping Cart Sequence

Basic Course

On the Shopping Cart Page, the Customer modifies the quantity of an Item in the Shopping Cart, and then presses the Update button.

The system stores the new quantity, and then computes and displays the new cost for that Item.

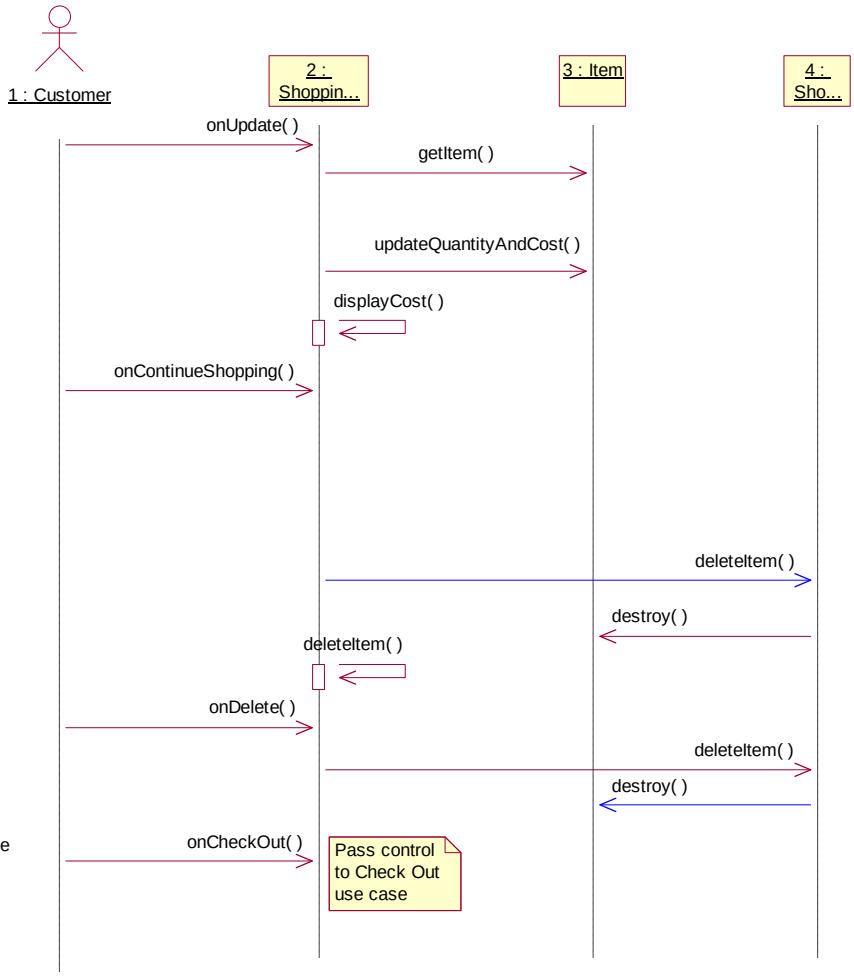
The Customer presses the Continue Shopping button. The system returns control to the use case from which it received control.

Alternate Courses

If the Customer changes the quantity of the Item to 0, the system deletes that Item from the Shopping Cart.

If the Customer presses the Delete button instead of the Update button, the system deletes that Item from the Shopping Cart.

If the Customer presses the Check Out button instead of the Continue Shopping button, the system passes control to the Check Out use case.



Use Case - Log In

Documentation:

Basic Course

The Customer clicks the Log In button on the Home Page. The system displays the Login Page. The Customer enters his or her user ID and password and then clicks the Log In button. The system validates the login information against the persistent Account data and then returns the Customer to the Home Page.

Alternate Courses

If the Customer clicks the New Account button on the Login Page, the system invokes the Open Account use case.

If the Customer clicks the Reminder Word button on the Login Page, the system displays the reminder word stored for that Customer, in a separate dialog box. When the Customer clicks the OK button, the system returns the Customer to the Login Page.

If the Customer enters a user ID that the system does not recognize, the system displays a message to that effect and prompts the Customer to either enter a different ID or click the New Account button.

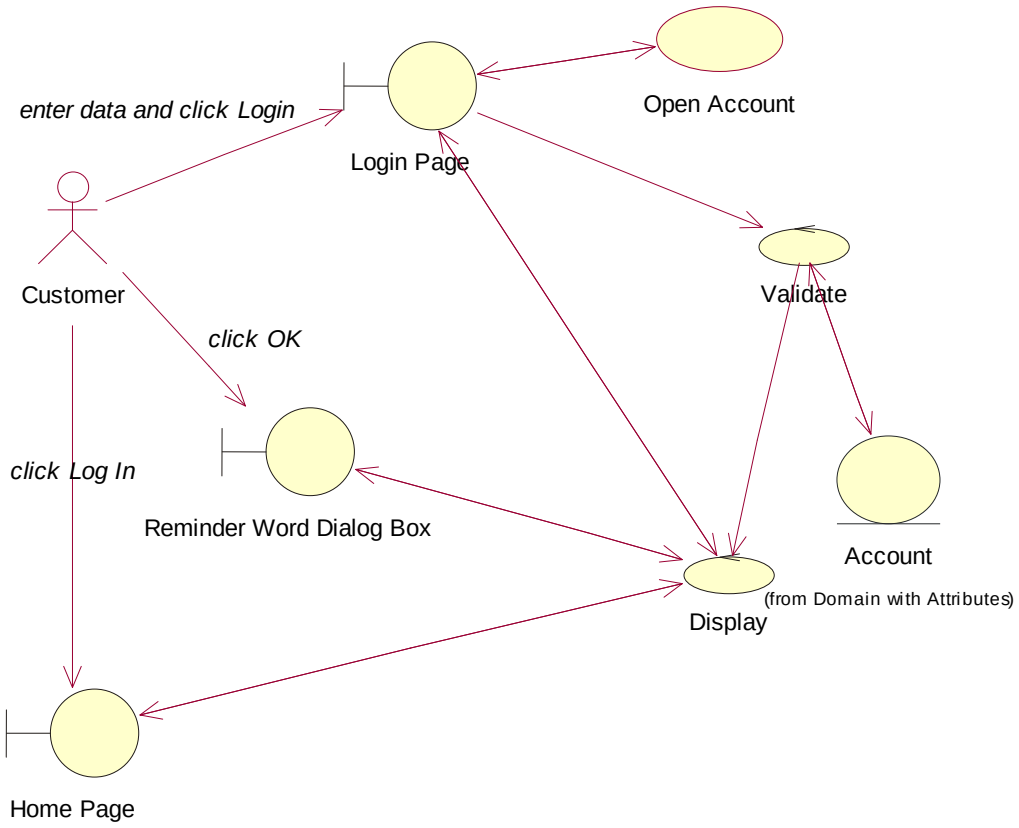
If the Customer enters an incorrect password, the system displays a message to that effect and prompts the Customer to reenter his or her password.

If the Customer enters an incorrect password three times, the system displays a page telling the Customer that he or she should contact customer service, and also freezes the Login Page.

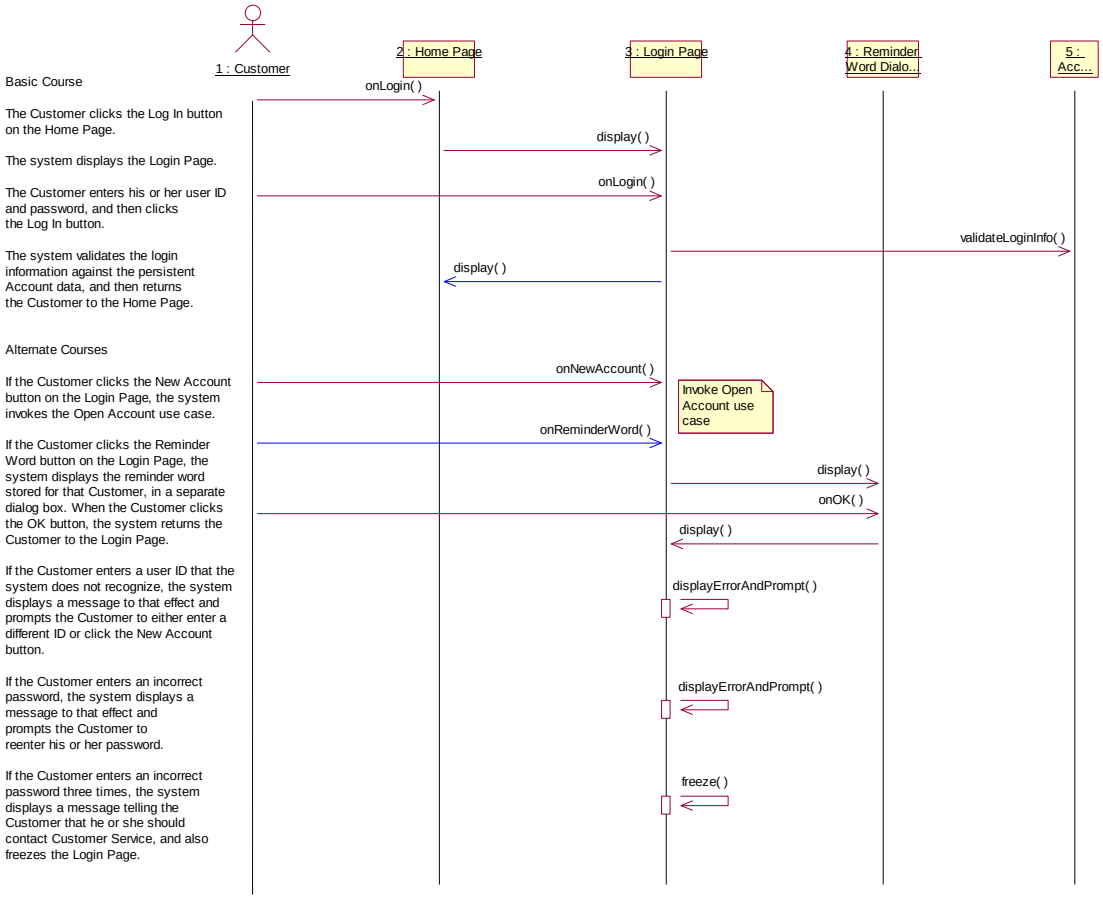
List of Associations

Customer Communicates with Log In.

Class Diagram - Log In Robustness



Interaction Diagram - Log In Sequence



Use Case - Open Account

Documentation:

Basic Course

The system displays the New Account Page. The Customer types his or her name, an e-mail address, and a password (twice), and then presses the Create Account button. The system ensures that the Customer has provided valid data and then adds an Account to the Master Account Table using that data. Then the system returns the Customer to the Home Page.

Alternate Courses

If the Customer did not provide a name, the system displays an error message to that effect and prompts the Customer to type a name.

If the Customer provided an email address that's not in the correct form, the system displays an error message to that effect and prompts the Customer to type a different address.

If the Customer provided a password that is too short, the system displays an error message to that effect and prompts the Customer to type a longer password.

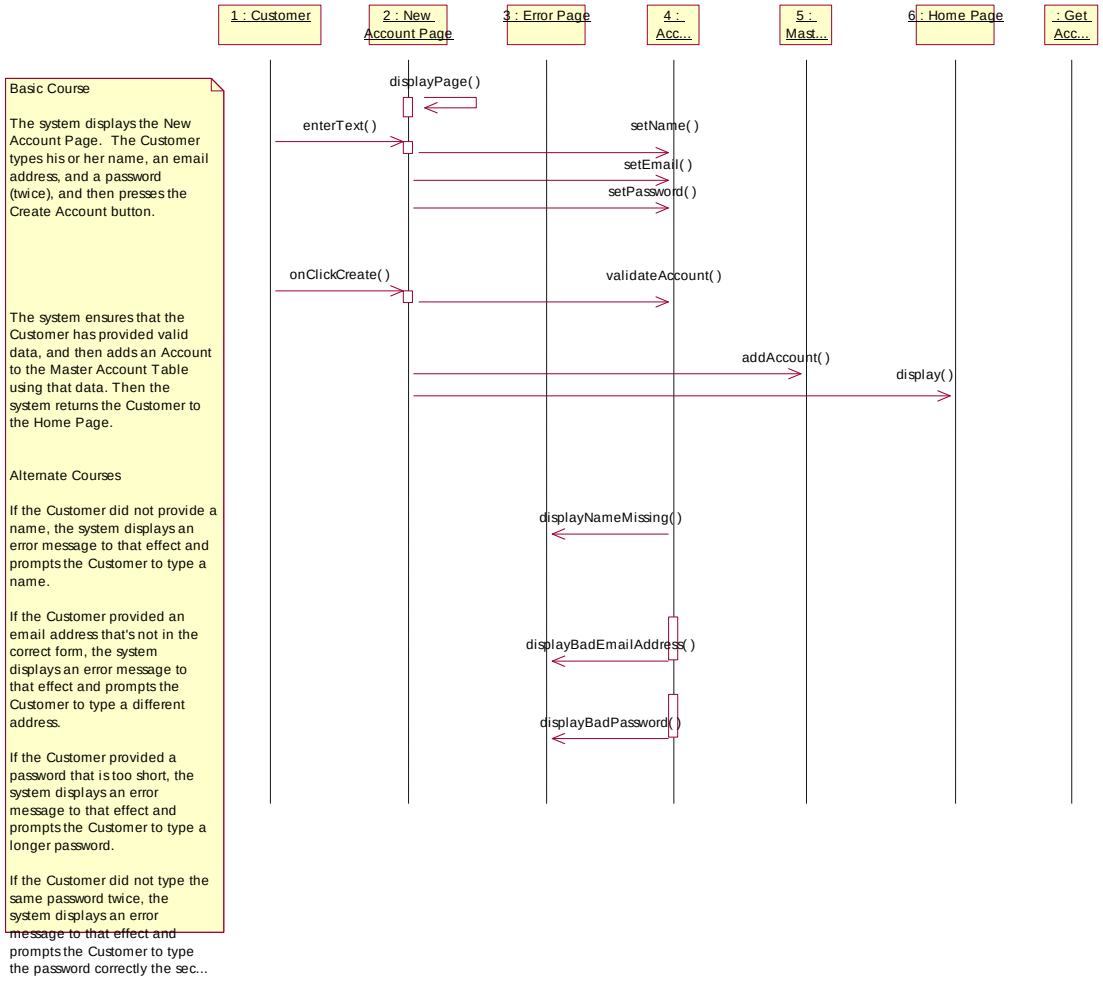
If the Customer did not type the same password twice, the system displays an error message to that effect and prompts the Customer to type the password correctly the second time.

If the account is already in the master account table, notify the user.

List of Associations

Customer Communicates with Open Account.
Login Page Communicates with Open Account.
Open Account Communicates with Login Page.

Interaction Diagram - Open Account Sequence Diagram



Use Case - Process Received Shipment

Documentation:

Basic Course

The Receiving Clerk ensures that the Line Items listed on the Purchase Order match the physical items. The Clerk waves the bar code on the packing slip under the sensor at the receiving station. The system changes the status of the Purchase Order to "fulfilled" and updates the quantity on hand values for the various Books. The Clerk hands the Books off to the Inventory Clerk.

Alternate Course

If the Receiving Clerk finds a mismatch between the Purchase Order and the physical items, the Clerk stops processing of the shipment until he or she is able to make a match.

List of Associations

Receiving Clerk Communicates with Process Received Shipment.
Process Received Shipment Communicates with Inventory Clerk.
Process Received Shipment Communicates with Receiving Station.

Use Case - Search by Author

Documentation:

Basic Course

The Customer types the name of an Author on the Search Page and then presses the Search button. The system ensures that the Customer typed a valid search phrase, and then searches the Catalog and retrieves all of the Books with which that Author is associated. The system retrieves the important details about each Book, and creates a Search Results object with that information.

Then the system displays the list of Books on the Search Results Page, with the Books listed in reverse chronological order by publication date. Each entry has a thumbnail of the Book's cover, the Book's title and authors, the average Rating, and an Add to Shopping Cart button. The Customer presses the Add to Shopping Cart button for a particular Book. The system passes control to the Add Item to Shopping Cart use case.

Alternate Courses

If the Customer did not type a search phrase before pressing the Search button, the system displays an error message to that effect and prompts the Customer to type a search phrase.

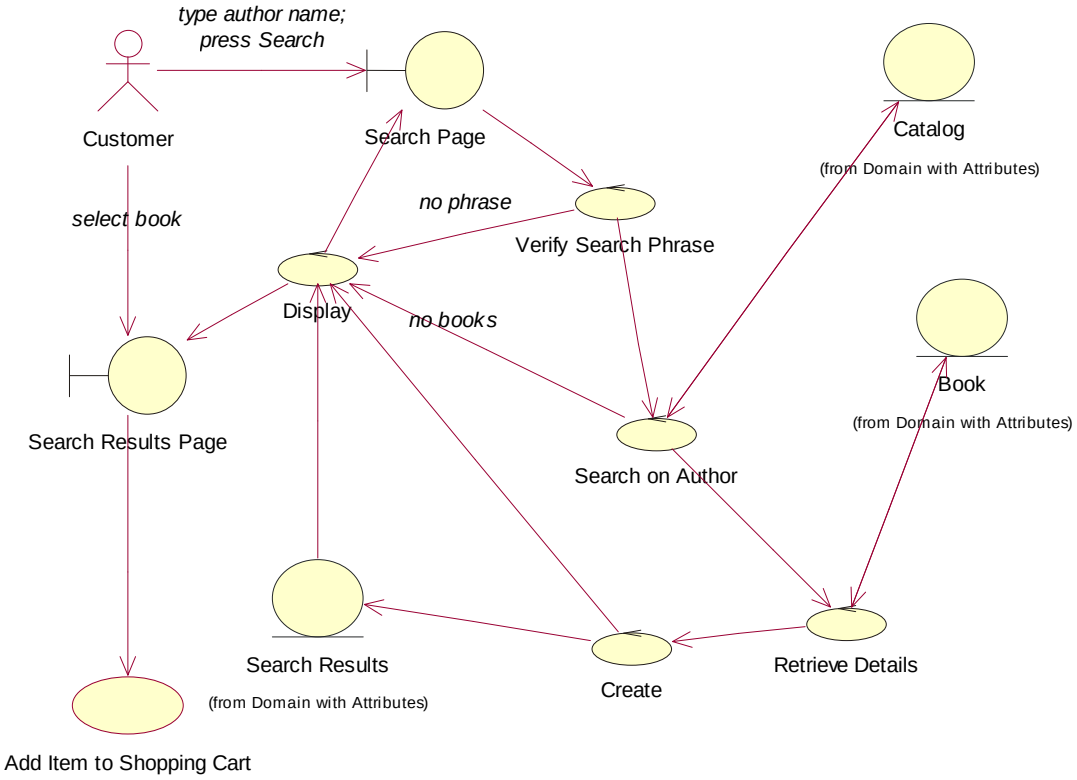
If the system was unable to find any Books associated with the Author that the Customer specified, the system displays a message to that effect and prompts the Customer to perform a different search.

If the Customer leaves the page in a way other than by pressing an Add to Shopping Cart button, the system returns control to the use case from which this use case received control.

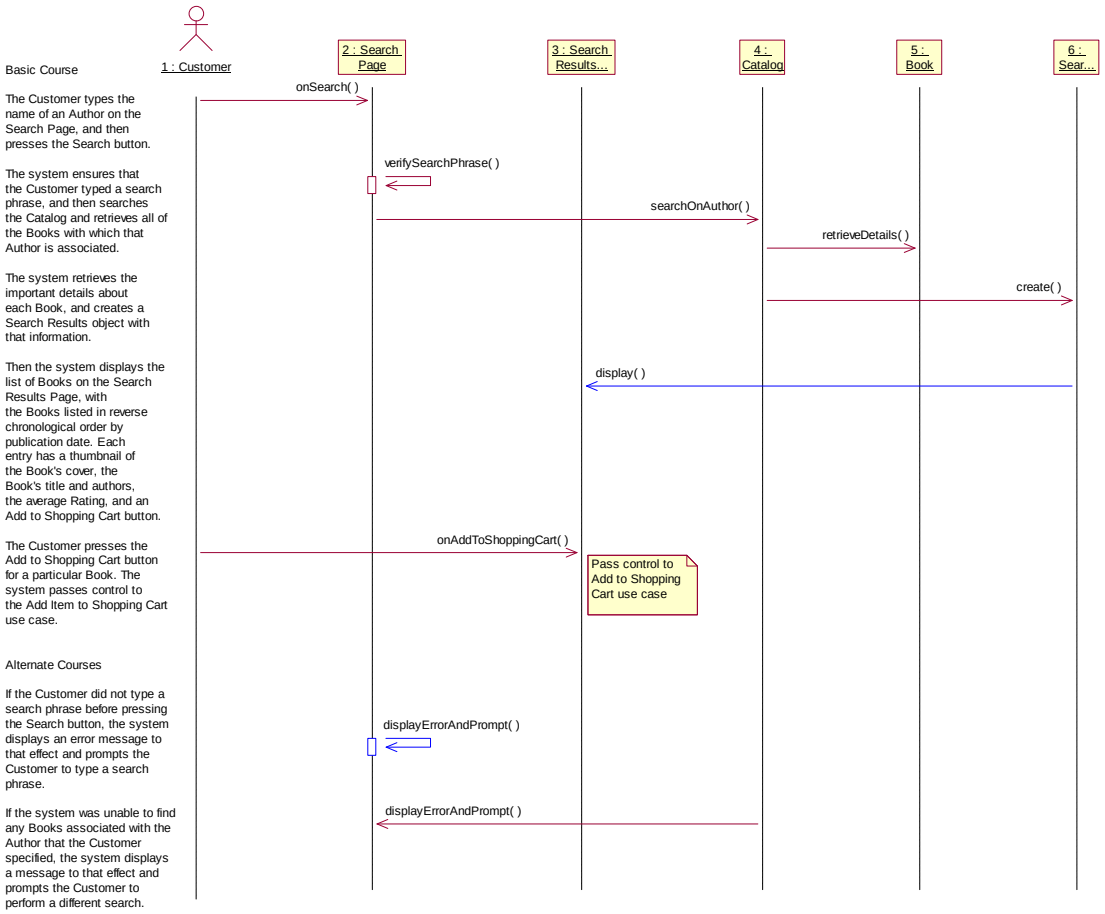
List of Associations

Customer Communicates with Search by Author.

Class Diagram - Search by Author Robustness



Interaction Diagram - Search by Author Sequence



Basic Course

The Customer types the name of an Author on the Search Page, and then presses the Search button.

The system ensures that the Customer typed a search phrase, and then searches the Catalog and retrieves all of the Books with which that Author is associated.

The system retrieves the important details about each Book, and creates a Search Results object with that information.

Then the system displays the list of Books on the Search Results Page, with the Books listed in reverse chronological order by publication date. Each entry has a thumbnail of the Book's cover, the Book's title and authors, the average Rating, and an Add to Shopping Cart button.

The Customer presses the Add to Shopping Cart button for a particular Book. The system passes control to the Add Item to Shopping Cart use case.

Alternate Courses

If the Customer did not type a search phrase before pressing the Search button, the system displays an error message to that effect and prompts the Customer to type a search phrase.

If the system was unable to find any Books associated with the Author that the Customer specified, the system displays a message to that effect and prompts the Customer to perform a different search.

If the Customer leaves the page in a way other than by pressing an Add to Shopping Cart button, the system returns control to the use case from which this use case received control.

Use Case - Ship Order

Documentation:

Basic Course

The Shipping Clerk ensures that the Items listed on the packing slip for the Order match the physical items. The Clerk waves the bar code on the packing slip under the sensor at the shipping station. The system changes the status of the Order to "shipping". Then the system retrieves the Shipping Method that the Customer specified for this Order and displays it on the Shipping Station Console. The Clerk weighs the set of physical items. The Clerk packages the Items. The Clerk attaches a manifest appropriate for the given shipping method. The Clerk waves the bar code on the manifest under the sensor. The Clerk sends the package out via the associated Shipper.

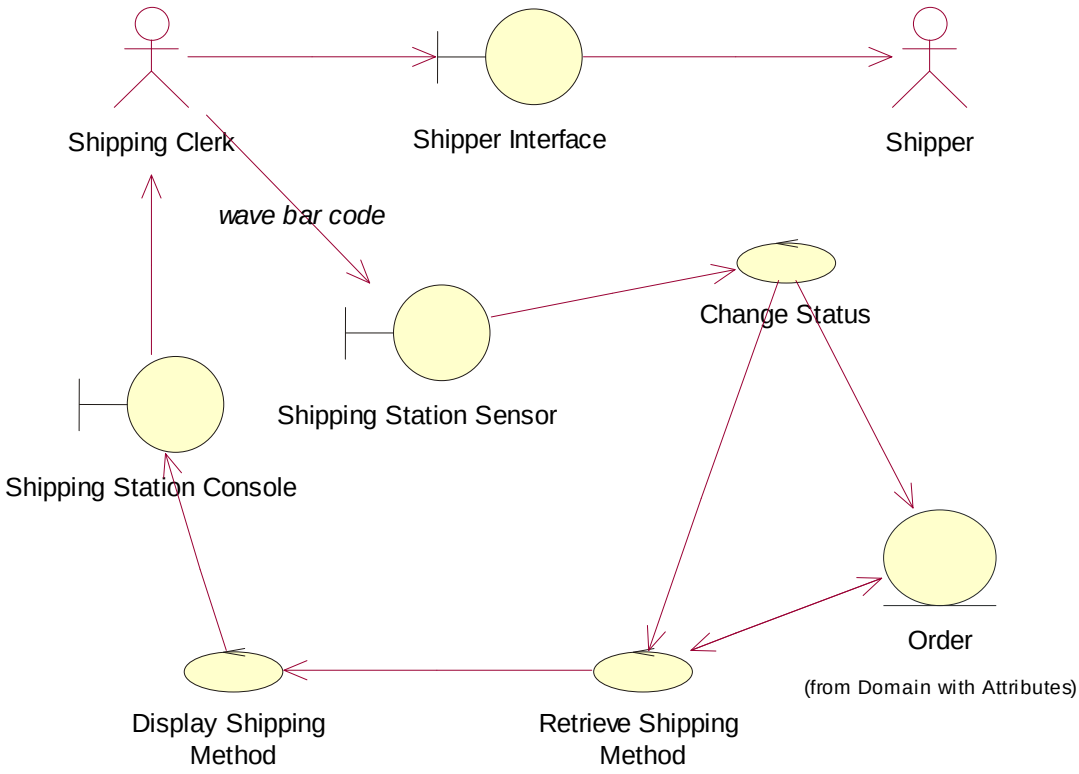
Alternate Course

If the Shipping Clerk finds a mismatch between the Order and the physical items, the Clerk stops processing of the Order until he or she is able to make a match.

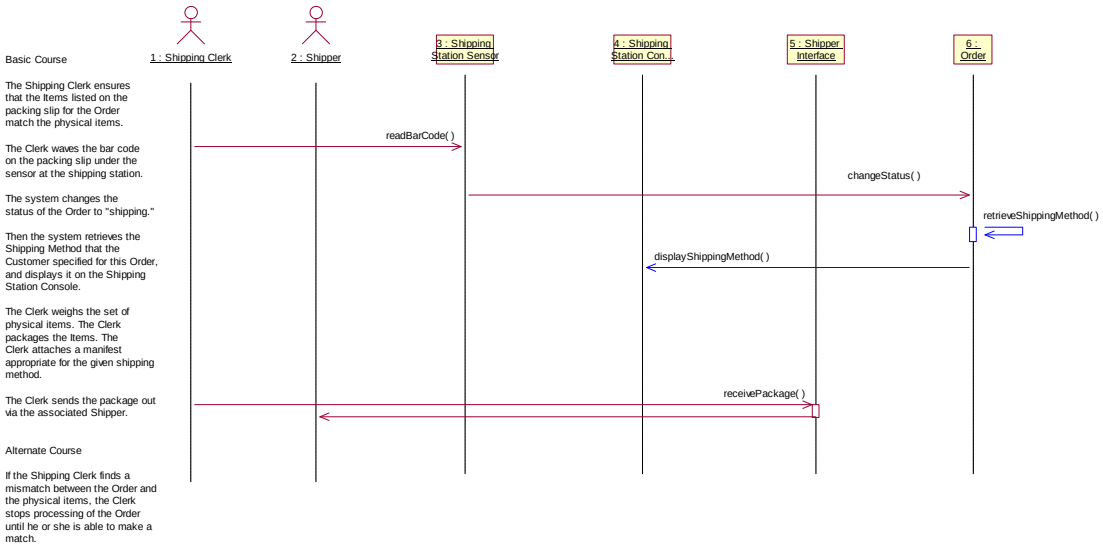
List of Associations

Shipping Clerk Communicates with Ship Order.
Ship Order Communicates with Shipper.
Ship Order Communicates with Shipping Station.

Class Diagram - Ship Order Robustness



Interaction Diagram - Ship Order Sequence



Use Case - Track Recent Orders

Documentation:

Basic Course

The system retrieves the Orders that the Customer has placed within the last 30 days and displays these Orders on the Order Tracking Page. Each entry has the Order ID (in the form of a link), the Order date, the Order status, the Order recipient, and the Shipping Method by which the Order was shipped.

The Customer clicks on a link. The system retrieves the relevant contents of the Order, and then displays this information, in view-only mode, on the Order Details Page. The Customer presses OK to return to the Order Tracking Page.

Once the Customer has finished viewing Orders, he or she clicks the Account Maintenance link on the Order Tracking Page. The system returns control to the invoking use case.

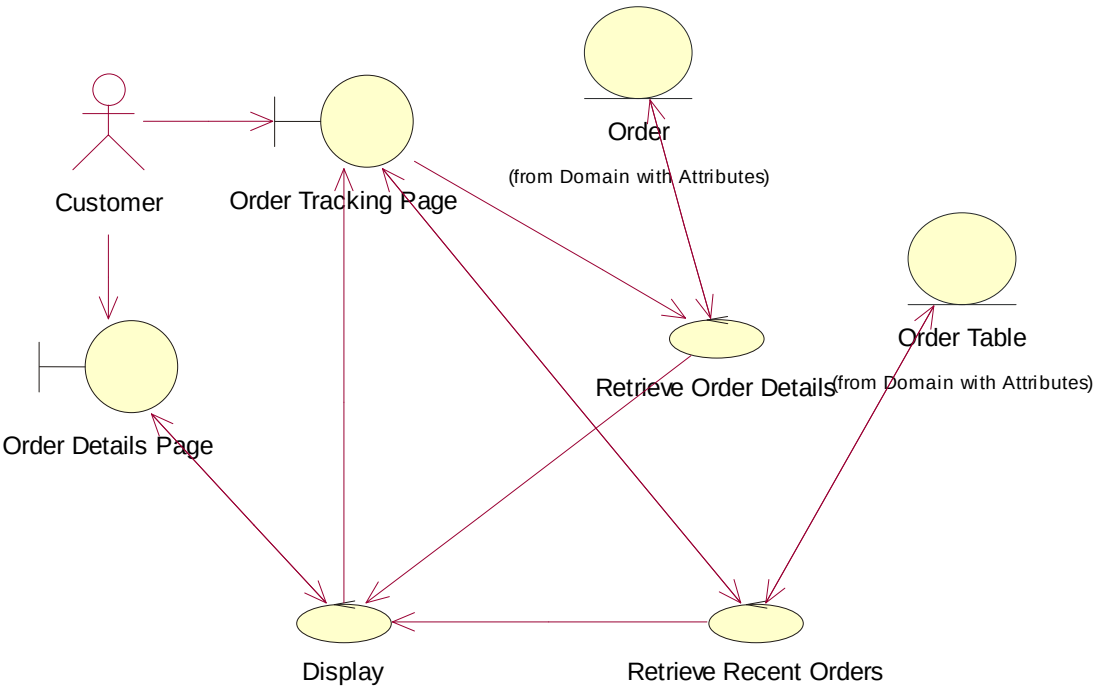
Alternate Course

If the Customer has not placed any Orders within the last 30 days, the system displays a message to that effect on the Order Tracking Page.

List of Associations

Customer Communicates with Track Recent Orders.

Class Diagram - Track Recent Orders Robustness



Interaction Diagram - Track Recent Orders Sequence

Basic Course

The system retrieves the Orders that the Customer has placed within the last 30 days, and displays these Orders on the Order Tracking Page. Each entry has the Order ID (in the form of a link), the Order date, the Order status, the Order recipient, and the Shipping Method by which the Order was shipped.

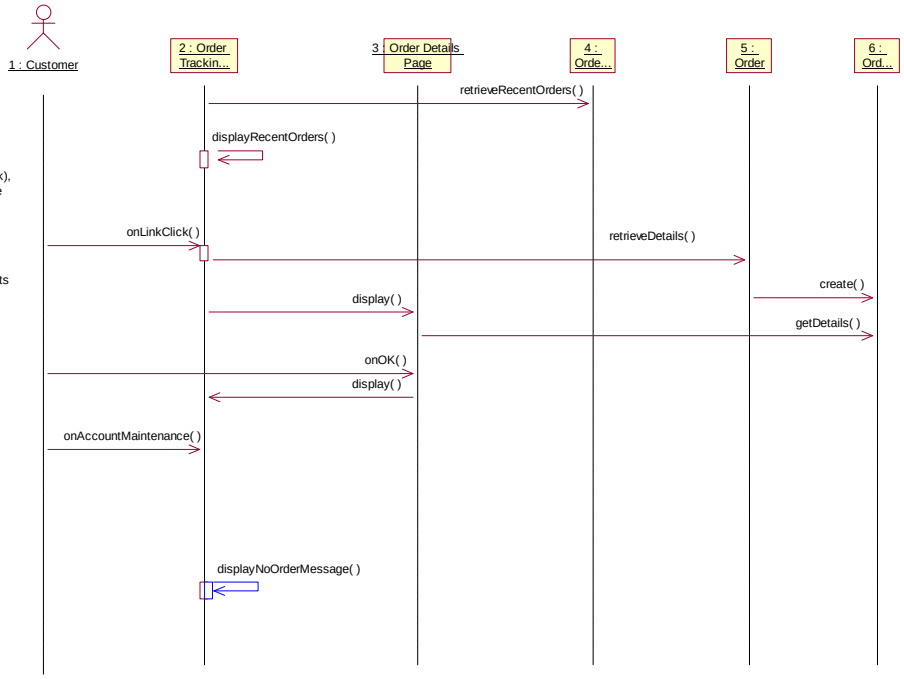
The Customer clicks on a link. The system retrieves the relevant contents of the Order, and then creates an Order Details object. The system displays the contents of this object, in view-only mode, on the Order Details Page.

The Customer presses OK to return to the Order Tracking Page.

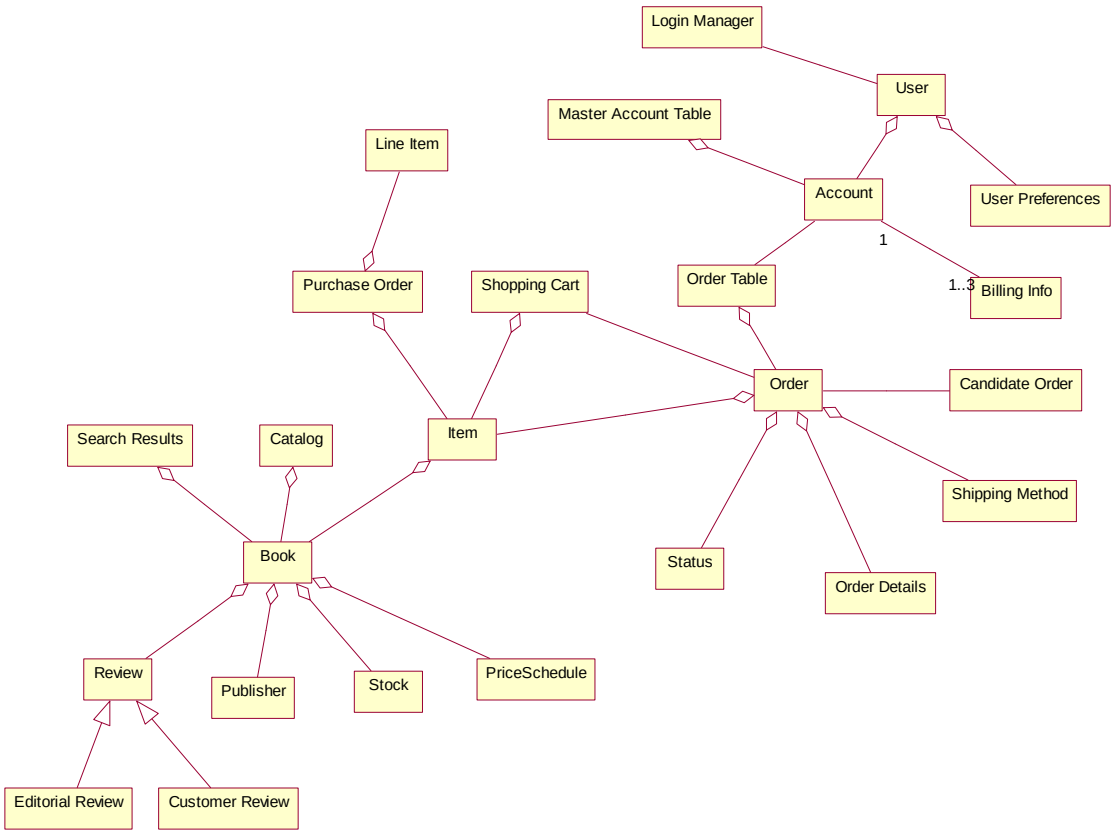
Once the Customer has finished viewing Orders, he or she clicks the Account Maintenance link. The system returns control to the invoking use case.

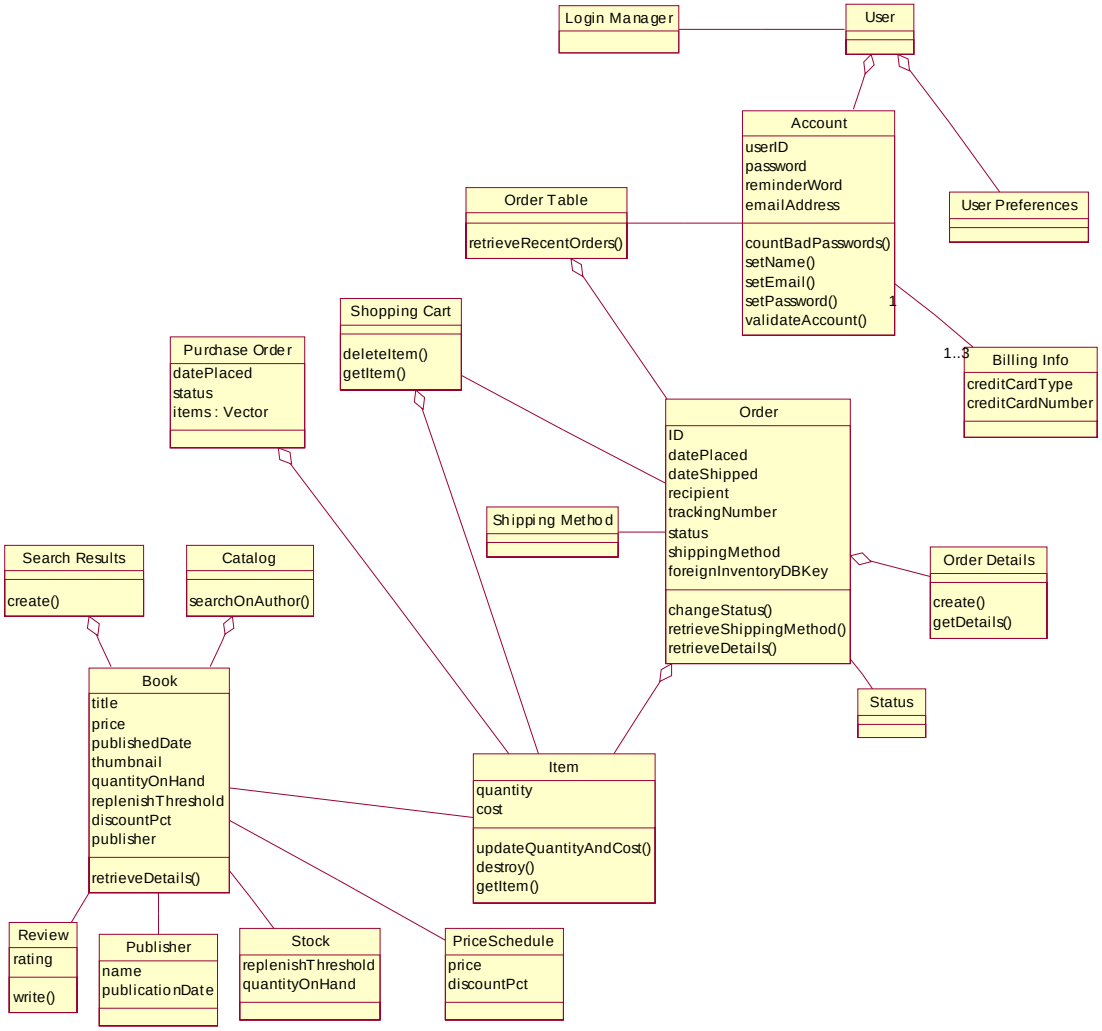
Alternate Course

If the Customer has not placed any Orders within the last 30 days, the system displays a message to that effect on the Order Tracking Page.

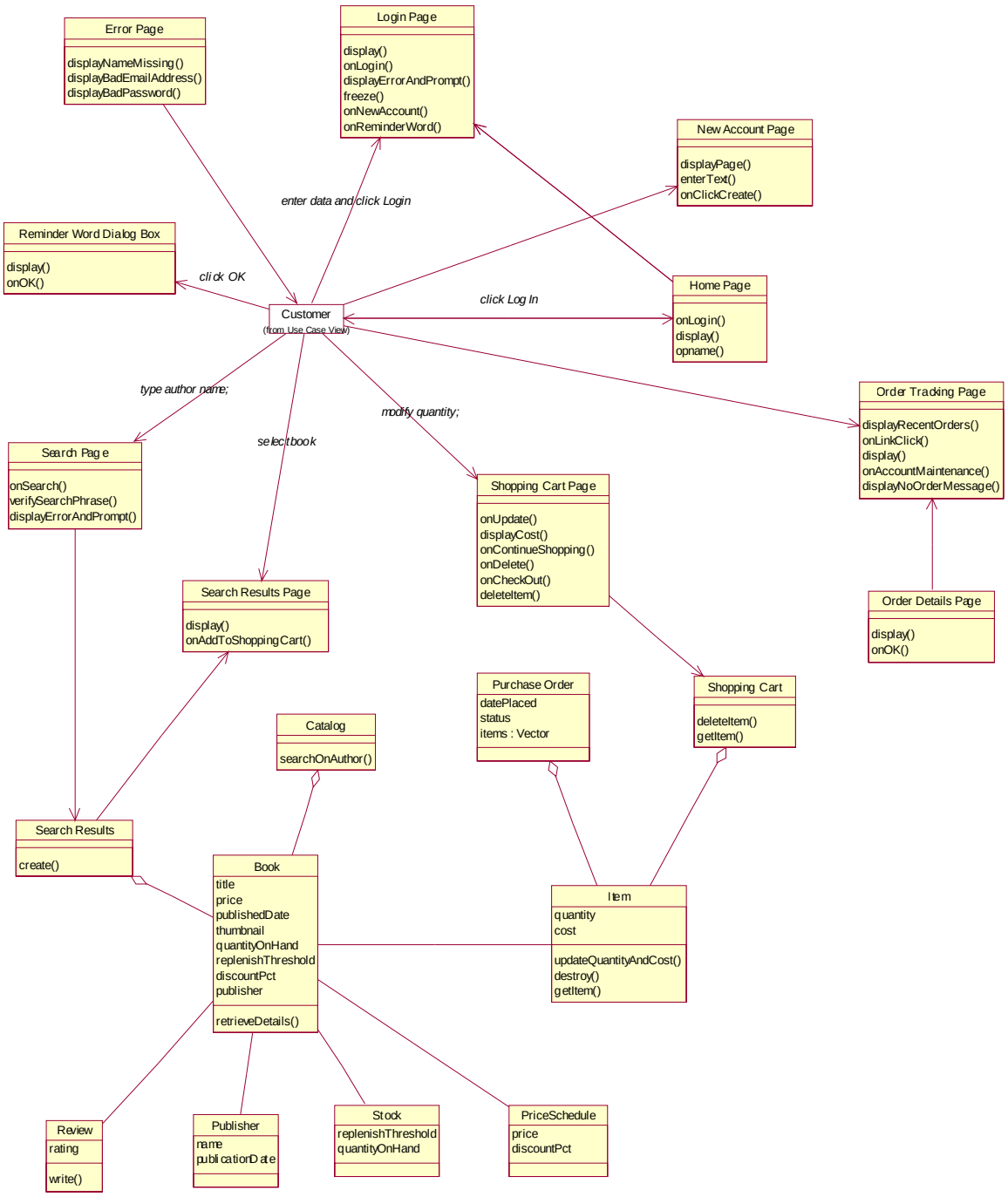


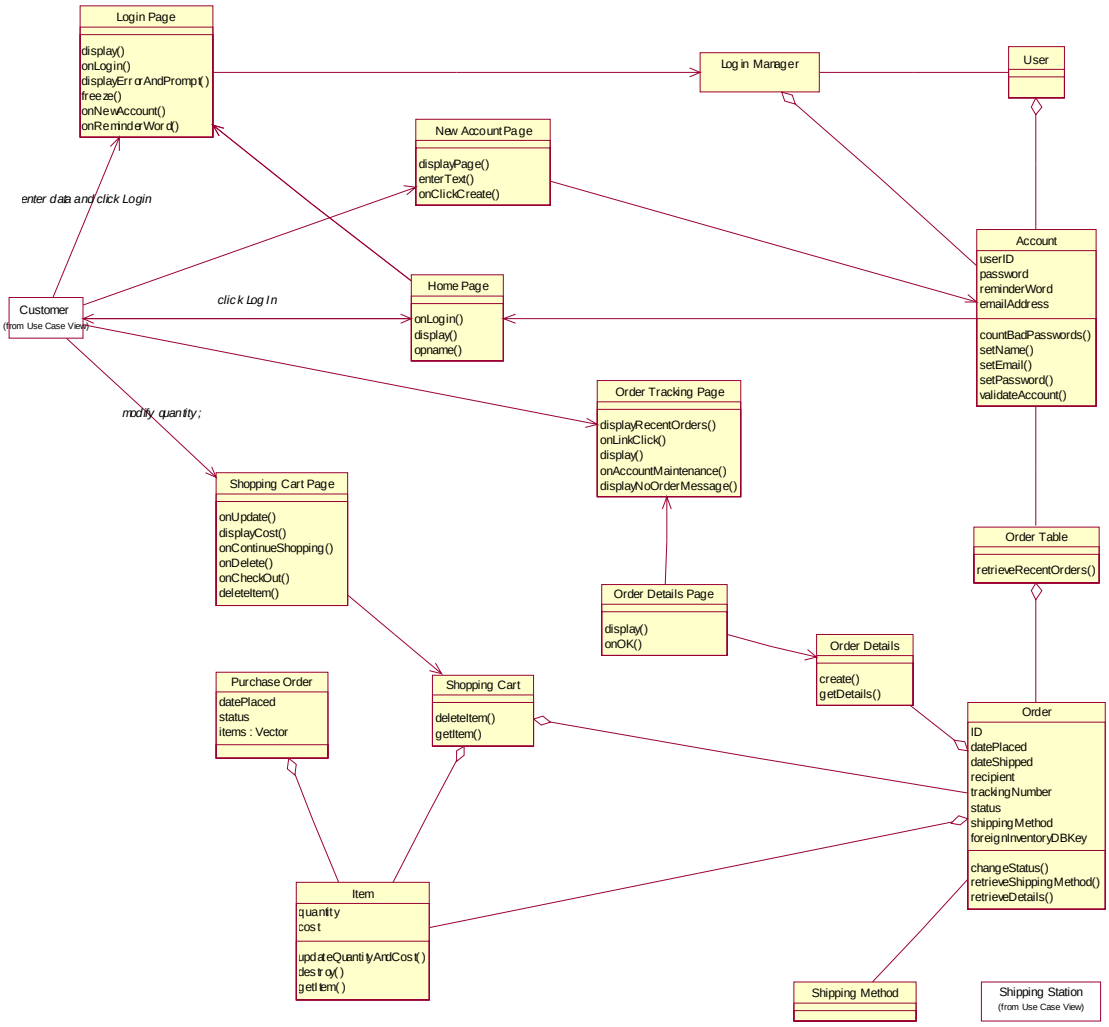
Domain Model

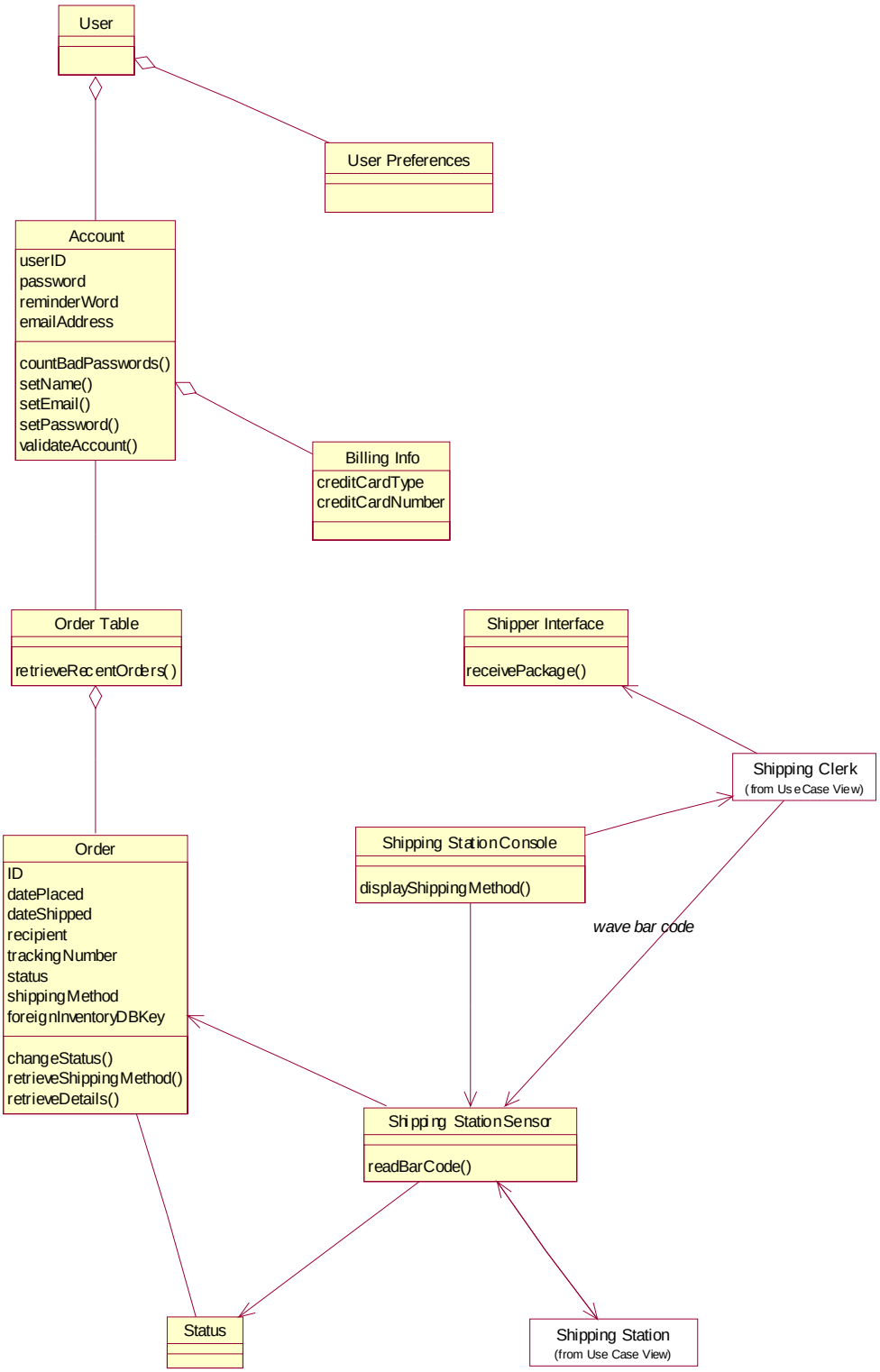



Login Manager
User
Account
userID
password
reminderWord
emailAddress
countBadPasswords()
setName()
setEmail()
setPassword()
validateAccount()
User Preferences
Order Table
retrieveRecentOrders()
1
1..3
Billing Info
creditCardType
creditCardNumber
Shopping Cart
deleteItem()
getItem()
Purchase Order
datePlaced
status
items : Vector
Order
ID
datePlaced
dateShipped
recipient
trackingNumber
status
shippingMethod
foreignInventoryDBKey
changeStatus()
retrieveShippingMethod()
retrieveDetails()
Shipping Method
Order Details
create()
getDetails()
Search Results
create()
Catalog
searchOnAuthor()
Status
Book
title
price
publishedDate
thumbnail
quantityOnHand
replenishThreshold
discountPct
publisher
retrieveDetails()
Item
quantity
cost
updateQuantityAndCost()
destroy()
getItem()
Review
rating
write()
Publisher
name
publicationDate
Stock
replenishThreshold
quantityOnHand
PriceSchedule
price
discountPct

Static Object Model




Login Page
display()
onLogin()
displayErrorAndPrompt()
freeze()
onNewAccount()
onReminderWord()
Login Manager
User
New Account Page
displayPage()
enterText()
onClickCreate()
enter data and click Login
Account
userID
password
reminderWord
emailAddress
countBadPasswords()
setName()
setEmail()
setPassword()
validateAccount()
Home Page
onLogin()
display()
opname()
click Log In
Customer
(from Use Case View)
Order Tracking Page
displayRecentOrders()
onLinkClick()
display()
onAccountMaintenance()
displayNoOrderMessage()
modify quantity;
Shopping Cart Page
onUpdate()
displayCost()
onContinueShopping()
onDelete()
onCheckOut()
deleteItem()
Order Table
retrieveRecentOrders()
Order Details Page
display()
onOK()
Order Details
create()
getDetails()
Purchase Order
datePlaced
status
items : Vector
Shopping Cart
deleteItem()
getItem()
Order
ID
datePlaced
dateShipped
recipient
trackingNumber
status
shippingMethod
foreignInventoryDBKey
changeStatus()
retrieveShippingMethod()
retrieveDetails()
Item
quantity
cost
updateQuantityAndCost()
destroy()
getItem()
Shipping Method
Shipping Station
(from Use Case View)


User
User Preferences
Account
userID
password
reminderWord
emailAddress
countBadPasswords()
setName()
setEmail()
setPassword()
validateAccount()
Billing Info
creditCardType
creditCardNumber
Order Table
retrieveRecentOrders()
Shipper Interface
receivePackage()
Shipping Clerk
(from UseCase View)
Shipping StationConsole
displayShippingMethod()
wave bar code
Order
ID
datePlaced
dateShipped
recipient
trackingNumber
status
shippingMethod
foreignInventoryDBKey
changeStatus()
retrieveShippingMethod()
retrieveDetails()
Shipping StationSensor
readBarCode()
Status
Shipping Station
(from Use Case View)

Appendix B – SDLC Roles

The following roles describe the personnel that are assumed to exist in Appendix C, The SDLC Outline. These roles are not necessarily assigned to a single person – one person may perform multiple roles, or a role may be split across multiple people where appropriate. The descriptions are informal because in any organization and even in different projects, there are different requirements and influences that dictate the actual role content and the precise job description. Prior to beginning a project, the roles must be defined precisely enough to prevent overlap or an incomplete work scope, with adjustments as necessary during a project.

Roles

- The **Project Sponsor** could be roughly described as the project champion. Generally someone either inside or outside of the organization who can acquire resources and support for the project and who also bears ultimate responsibility for its success or failure.

- The **Chief Information Officer** is responsible for the overall control of the data processing enterprise and while generally not directly involved in a project, has control of many of the resources needed and may have technical review of all projects and resource allocations.

- The **Client** is the person, group or organization that will be the primary user of the software product. The **Users** role is part of the **Client** role.

- The **Client Liaison** is the person responsible for interfacing with the client for the software. This is an important task because project success is often dependent more on the client's view of the product than the technical results. This person is responsible for much of the *requirements development* of the project, ongoing demonstrations of the product, and change orders in both directions and scheduling.

- The **Configuration Lead** is responsible for maintaining the repository for all materials associated with the project. While this is primarily a matter of maintaining a version control system for development and QA, it vital that all materials be available to all members of the team as needed. The ideal solution is a web site for the information, but it should be protected from outside viewing. Some parts of the project may need to have limited access, such as the project planning and reports, and a variety of data formats may be needed.

- The **Documentation Lead** is responsible for leading the technical writers in developing the documentation for the product and the users, and possibly for portions of the training.

- The **Maintenance Team** is responsible for maintaining the product after implementation. They are not a major part of the SDLC but do need to be recognized. Often, these people are part of the **Project Team**.

- The **Procurement Officer** is responsible for obtaining resources for the project and supports the **Project Manager** in handling resource allocation.

- The **Project Team** is everyone in the project that doesn't have a particular role at any given time. They may be software developers or testers, designers; systems support personnel or anyone else that needs to be considered in completing the project.

- The **Project Manager** is ultimately responsible for the overall progress of the project and for reporting to the Project Sponsor as needed.

- The **Quality Assurance Lead** is responsible for the development of the testing plans and for insuring that all testing is properly done, possibly through the maintenance cycle.

- The **Requirements Lead** is responsible for collecting requirements information and organizing it for the design team. This person may need more technical expertise than the **Client Liaison**, depending on the personnel available.

- The **Security Lead** is responsible for the security aspects of the software and the implementation, possible through the maintenance cycle.

- The **System Architect** is responsible for the design phase of the project.

- The **Integration Lead** is responsible for developing the infrastructure and support tools for the software and for the configuration and execution of the system builds. This person is also responsible for interfacing with the system support personnel as part of the process of insuring that computers for development and testing are available and properly configured.

- The **Trainer Lead** is responsible for organizing and executing the training program for the **Users**.

- The **Users** are the people who will actually use the product and the documentation and will be the recipients of training. The **Users** role is part of the **Client** role.

Appendix C – The SDLC Outline

The following is an outline of the tasks, deliverables and roles for the SDLC.

Software Development Life Cycle Process

Starting Condition

IT nomination request received or expected.

Deliverables

- Installed software.
- Defect Management System.
- User Training.
- Documentation.
- Customer support plan.
- SDLC Performance Report.

Roles

See Appendix B.

Planning Phase

Starting Condition:

IT nomination request received or expected.

Objectives:

Identify a need to enhance business practice through a software development project, determine the assumptions and constraints on the solutions and propose a solution.

Tasks and Activities:

Requirements development and management

- Develop the project proposal
- Identify project stakeholders
- Develop a Project Boundary Document
- Develop the Concept of Operations document

Project management

- Identify project sponsor and project manager
- Develop a role statement
- Create a preliminary plan and schedule
- Develop a risk assessment statement
- Develop a Benefit/Cost Analysis
- Conduct the Phase Review

Software design

Create the Use Case business model.

Software development

Project implementation

Roles and Responsibilities:

- Sponsor - responsible for ensuring that resources for the project will be available and for identifying underlying assumptions and integration with the overall business process. If primarily responsible for identifying project stakeholders and the benefit side of the Benefit/Cost Analysis.
- Project Manager - responsible for the technical aspects of the project including identifying the necessary roles, plans and schedules, and identifying project risk factors.
- Integration Lead - develop the Integration Plan and proceed with the acquisition of the necessary hardware and software resources for the project and provide a cost estimate.

Deliverables:

- Project Proposal
- Concept of Operations Document
- Use Case Business Model
- Role Statement
- Project Plan
- Project Boundary Document
- Benefit/Coat Analysis

- Risk Assessment
- IT Nomination Packet
- Measurement Plan
- Conduct the Phase Review
- Project Audit Report

Issues for Consideration:

Phase Review Activity:

- Get approval for the Project Proposal as needed.
- Collect metrics regarding the time and other resources required for various parts of this phase.
- Update the Project Audit Report continuously

System Analysis Phase

Starting Condition:

Decision is made to proceed with the project.

Objectives:

Determine the functional design of the system with respect to inputs, outputs, processes and interfaces and develop baseline plans for proceeding.

Tasks and Activities:

Requirements development and management

- Develop the Requirements Document
- Develop a System Security Plan
- Develop a Quality Assurance Plan
- Develop an interface control document

Project management

- Develop a project workscope.
- Develop a baseline project plan and timeline.
- Develop the risk management plan.
- Develop a security and privacy plan.
- Create a defect tracking system.
- Select the project team.
- Begin the project costing document
- Conduct the Phase Review.

Software design
Software development
Project implementation

Roles and Responsibilities:

- Sponsor - Insure that resources for the improved plan are available and that costs are within acceptable limits. Review and approve integration between the client and the project team.
- Project Manager - responsible for directing the development of the project plan and assigning roles to staff members. Responsible for the accuracy and completeness of all deliverables, and for reporting on project progress.
- Project Team - Complete assigned tasks and contribute expertise in planning the project and evaluating technical risk factors.
- Requirements Lead - Plan and conduct the collection of the data and the formation of the Requirements Document. This will include considerable work with clients identifying the appropriate means for data collection and the vetting of the final document.
- System Architect - Work with the Requirements Lead and the other Project Team members to insure that the requirements can be met within the available resources.

- Security Lead - Insure that security and privacy concerns are identified and accounted for in the planning process.
- Chief Information Officer - Approval of project for continuance.
- Client Liaison - interface with the client to insure that the project meets requirements by getting approval for the Requirements Document.
- Configuration Lead - insure that all documents are properly archived and protected as necessary.
- Integration Lead - insure that the hardware and software system are available and properly configured. Work with the Project Team to solve problems.

Deliverables:

- Requirements Document
- Security Plan
- Interface Control Document
- Use Case Model
- Workscope
- Baseline Project Timeline
- Baseline Project Task list
- Risk Management Plan
- Quality Assurance Plan
- Defect Tracking System
- Configuration Management Plan
- Project Plan
- Master Test Plan
- Conduct the Phase Review
- Project Audit Report

Issues for Consideration:

- If sensitive data is involved, determine the appropriate standards for privacy and insure that consideration is given to this issue. Insure that all staff have the appropriate levels of access. Insure that the planned implementation platforms can enforce the privacy standards.
- Give appropriate attention to the possibility of using an existing system or purchasing software to meet these needs if the goals can be met and the cost is advantageous.
- The Interface Control Document should identify all systems to which this software will interface (software and otherwise) and determine how they will interface with regard to business processes and data exchange. At this point, the details may not be complete, but it is vital that all interactions be known and understood.

Phase Review Activity:

- Does the project, as planned, meet the stated goals and is it feasible technically and from a resource point-of-view.

- Collect metrics regarding the time and other resources required for various parts of this phase.
- Get approval to proceed.
- Update the Project Audit Report continuously.

System Design Phase

Starting Condition:

Requirements document has reached a stable state, the plans covering project content are in place, and planning has proceeded to the point that tasks can be assigned.

Objectives:

Transform the system requirements into detailed specifications that describe how the system is to meet the functional, physical, interface and data requirements. This process may be iterative, producing several models.

Tasks and Activities:

Requirements development and management

- Develop the system specification
- Update requirements

Project management

- Update the project plan
- Conduct the Phase Review

Software design

- Develop the Analysis Model
- Develop the user interface storyboards
- Develop change orders

Software development

Project implementation

- Develop a security plan
- Develop a system test plan
- Develop an acceptance test plan
- Develop the acceptance test cases
- Develop the documentation plan
- Develop a user support plan
- Develop a training plan
- Develop an integration plan
- Develop a conversion plan
- Develop a deployment plan

Roles and Responsibilities:

- Project Manager - leadership role in the staffing and development of the specification and design. Must review all deliverables for accuracy and approve. Responsible for interfacing with the Sponsor to insure that any changes in cost or resources are approved.
- Project Team - responsible for task assignments.

- Requirements Lead - responsible for insuring that the System Specification accurately reflects the Requirements Document and get client approval of any changes.
- System Architect - responsible for the overall design process, for risk assessment and for the technical properties of the system and acceptance tests.
- Procurement Officer - responsible for acquiring resources and contract management.
- Client Liaison - interface with the client to insure that the project meets requirements by doing necessary walkthroughs of design documents (specifications and user interface storyboards) with the client and getting approval. For any changes, produce change orders in collaboration with the Project Manager.
- Configuration Lead - insure that all documents are properly archived and protected as necessary.
- Integration Lead - insure that the hardware and software system are available and properly configured. Work with the Project Team to solve problems.
- Deployment Lead - develop a deployment plan and get approval.

Deliverables:

- System Specification
- Software Analysis Model
- User Interface Storyboards
- Acceptance Test Plan
- System Test Plan
- Acceptance Test Cases
- Change Order
- Documentation Plan
- User support Plan
- Training Plan
- Integration Plan
- Conversion Plan
- Deployment Plan
- Conduct the Phase Review
- Project Audit Report I
- Issues for Consideration:

I**ssues for Consideration:**

Look for opportunities to use off-the-shelf (OTS) components, where practical, in assessing the design. Pay particular attention to implementation decisions, such a language and platform, and how they affect the design.

Phase Review Activity:

- The design phase is the best opportunity for the project team to identify technological risk and its potential impact on the future of the project. It is particularly important that all high-risk components be identified and maximum delays and costs be properly inserted in the project plan and project report.
- Insure that client approval of the specification and user interface is gained before proceeding.
- Reassess cost and resource needs based on the updated project plan and get approval for continuance.
- Collect metrics regarding the time and other resources required for various parts of this phase.
- Review the overall handling of the phase to make recommendations for improvements in future projects.

Update the Project Audit Report continuously.

Technical Design Phase

Starting Condition:

The System Analysis Model has reached a state of detail that allows all or part of the Technical Design to proceed.

Objectives:

Transform the system design into a language specific system specification that will meet the system requirements for functional, physical, interface and data requirements. This process may be iterative and create multiple designs. Establish plans for the development and testing of the software.

Tasks:

Requirements development and management

- Update the requirements document.
- Update the system specification document.
- Project management.

Project management

- Update the project plan.
- Conduct the Phase Review.

Software design

- Develop the Design Model.
- Develop the data model specification.
- Develop the user interface mockups.
- Develop and process change orders.

Software development

- Develop a unit test plan.
- Develop a software standards document.

Project implementation

- Update the system test plan.
- Create a system operation plan.
- Create a configuration management system.
- Update the acceptance test plan.
- Develop a contingency plan.

Roles and Responsibilities:

- Project Manager - leadership role in the staffing and development of the advanced design. Must review all deliverables for accuracy and approve. Responsible for interfacing with the Sponsor to insure that any changes in cost or resources are approved.
- Project Team - responsible for task assignments.
- Requirements Lead - responsible for insuring that the design accurately reflects the Requirements Document and get client approval of any changes.
- System Architect - responsible for the overall design process, and for the quality of the design. Responsible for the programming standards document and the unit test plan, system and acceptance tests.

- Configuration Lead - with the System Architect, develop the configuration management system.
- Procurement Officer - responsible for acquiring resources and contract management.
- Client Liaison - interface with the client to insure that the project meets requirements by doing necessary walkthroughs of design documents (specifications and user interface storyboards) and getting approval. For any changes, produce change orders in collaboration with the Project Manager.
- Configuration Lead - insure that all documents are properly archived and protected as necessary.
- Integration Lead - insure that the hardware and software system are available and properly configured. Work with the Project Team to solve problems. Develop or acquire software for installing and managing the deployed system during the development cycle and create the System Administration Plan.

Deliverables:

- Software Design Model
- Data Model Specification
- User Interface Mock-ups
- Unit Test Plan
- Change Order
- Programming Standards
- Development Contingency Plan
- Conduct the Phase Review
- Project Audit Report

Issues for Consideration:

- The team must be careful to maintain a design and not an implementation at this phase. This is an opportunity to review the design in detail and find any problems before they creep into the development.
- During this phase, there is an opportunity to begin development efforts on foundational parts of the design that are necessary for the larger project scope, and to work on high risk areas of the design.
- Because many security issues are programming language dependent, a security analysis at this stage and an update of the security plan is important. Security issues are a fundamental part of the programming standards document.

Phase Review Activity:

- As with the previous phase, evaluate for high risk components or areas of the implementations that have functionality that is not known by the design team.

- Insure that any changes in the Specification and user interface are reported to the client and approval is gained before proceeding.
- Reassess cost and resource needs based on the updated project plan and get approval for continuance.
- Collect metrics regarding the time and other resources required for various parts of this phase.
- Review the overall handling of the phase to make recommendations for improvements in future projects.

Development Phase

Starting Condition:

The Technical Design has progressed to the point where development tasks can be assigned, tested and monitored.

Objectives:

Create and test working software that meets the specifications for the system.

Tasks:

Requirements development and management

- Update the requirements document
- Update the specification document
- Project management

Project management

- Update the project plan
- Conduct the Phase Review

Software design

- Update the system design
- Develop and process change orders

Software development

- Develop the running software
- Perform successful unit tests
- Defect list

Project implementation

- Develop the user documentation
- Develop the product documentation
- Develop the training courseware
- Develop the operations system and documentation
- Perform successful system tests

Roles and Responsibilities:

- Project Manager - leadership role in the staffing and direction of the development team. Must review all deliverables for accuracy and approve. Responsible for interfacing with the Sponsor to insure that any changes in cost or resources are approved.
- Quality Assurance Officer - tests the product according to the system test plan and reports any defects to the development team.
- Project Team - responsible for task assignments and for unit tests. Team as a whole is responsible for smoke testing the software to insure that it can past to quality assurance.
- Requirements Lead - insure that the developed software accurately reflects the Requirements Document and get client approval of any changes. Work closely with the Client Liaison to develop and execute a schedule of demonstrations for the client. For any changes, produce change orders in collaboration with the Project Manager.
- Documentation Lead - develop the documentation and verify through the project team with approval by the Project Manager and System Architect.

Work with the Client Liaison to get customer approval of the documentation format.

- System Architect - responsible for the overall development process, and for the quality of the software. Responsible for insuring that unit tests are properly completed. Responsible for product documentation.
- Configuration Lead - work with the System Architect and the development team to build and configure the product and to develop the operational tools and documentation.
- Procurement Officer - responsible for acquiring resources and contract management.
- Client Liaison - provide the client with demonstrations of the product at regular intervals and obtain approval for the implementation or identify areas that require modification. For any changes, produce change orders in collaboration with the Project Manager.
- Configuration Lead - insure that all documents are properly archived and protected as necessary. Work with the development team and the Integration Lead to insure that the software versioning works properly and effectively for development and quality assurance.
- Integration Lead - insure that the hardware and software system are available and properly configured. Work with the Project Team to solve problems. Insure that system builds operate properly and update the System Administration and Deployment plans as necessary.

Deliverables:

- Functional software
- Documentation
- Successful System Test
- Change Order
- Training courseware
- System administration plan
- Conduct the Phase Review
- Project Audit Report

Issues for Consideration:

Software leaving development often has known defects that are minor or are accepted by the client for the short term. Because this phase often represents a majority of the time, the collection of metrics regarding the time and other resources required for is particularly important.

Phase Review Activity:

- Insure that the Requirements Document, Specification and system documentation are updated to reflect any changes.
- Collect metrics regarding the time and other resources required for this phase.
- Review the overall handling of the phase to make recommendations for improvements in future projects.
- Update the Project Audit Report continuously.

Implementation Phase

Starting Condition:

All or part of the developed software is ready for installation and testing.

Objectives:

Install, operate and test the system to insure that it meets the established system specifications, that all subsystems meet specification and that all interfacing systems operate correctly. Specifications include user satisfaction, documentation acceptance and successful training.

Tasks and Activities:

Requirements development and management

- Review and update the requirements document
- Review and update the specification document

Project management

- Conduct the Phase Review

Software design

- Develop and process change orders

Software development

- Update product
- Quality assurance testing

Project implementation

- Install and test software
- Test and finalize the system operation
- Perform system integration
- Collect user feedback
- Perform the acceptance test
- Revise the user support plan
- Perform training
- Execute a user survey

Roles and Responsibilities:

- Project Manager - leadership role directing the implementation team. Must review all deliverables for accuracy and approve. Must interface with the Sponsor for any issues that arise with the client.
- Quality Assurance Officer - continue product testing.
- Project Team - responsible for task assignments.
- Configuration Lead - test and finalize the operational system.
- Procurement Officer - responsible for acquiring resources and contract management.
- Procurement Officer - responsible for acquiring resources and contract management.
- Users - actively work with the product during training and system testing and report any potential defects.
- Client Liaison - interface with the client to insure that the project meets requirements by being actively involved in user testing and handling insertion of defects. Responsible for the acceptance test, client training

management and the user survey. For any changes, produce change orders in collaboration with the Project Manager.

- Configuration Lead - insure that all documents are properly archived and protected as necessary.
- Deployment Lead - oversee the deployment process.
- Integration Lead - insure that the hardware and software system are available and properly configured. Work with the Project Team and users to solve operational problems.
- Maintenance Team - develop a plan for maintaining the product and integrating with the defect management system.

Deliverables:

- Deployed software
- Required training
- Successful implementation report
- Training survey
- User response surveys
- Completed acceptance test
- Change orders
- Maintenance plan
- Conduct the Phase Review
- Project Audit Report

Issues for Consideration:

Insure that any security testing is accomplished at this stage.

Phase Review Activity:

- Insure that the Requirements Document, Specification and system documentation are updated to reflect any changes.
- Collect metrics regarding the time and other resources required for this phase.
- Review the overall handling of the phase to make recommendations for improvements in future projects.
- Update the Project Audit Report continuously.

Conclusion Phase

Starting Condition:

All testing is complete and accepted by the customer. All training is complete and all deliverables are complete or a completion date can be established.

Objectives:

Transition the software to the maintenance cycle and conduct a review of the project for the purpose of improving the software development life cycle.

Tasks:

Requirements development and management

Project management

- Create a maintenance plan
- Conduct the project review
- Update policies and procedures.

Roles and Responsibilities:

- Project Manager - leadership role directing the move to maintenance mode. Must review all deliverables for accuracy and approve. Provide the interface to the maintenance team.
- Quality Assurance Lead - continue product testing.
- Project Team - responsible for task assignments.
- Maintenance Team - begin maintenance of the product, including the handling of defects, if so required.
- Procurement Officer - responsible for acquiring resources and contract management.
- Client Liaison - Continue to assure client satisfaction and interface with the Defect Management System and the Maintenance Team.
- Configuration Lead - insure that all documents are properly archived and protected as necessary. Prepare the project repository for closing.

Deliverables:

- Final customer approval
- Analysis of customer satisfaction
- Analysis of project performance
- Conduct the Phase Review
- Project Audit Report

Issues for Consideration:

Project Review Activity:

Evaluate the entire project for ways to improve the process at each phase and to establish improved metrics for estimating projects.

Appendix D – SDLC Form List

The following is a list of all forms referenced in the SDLC outline.

Acceptance Test Case
Acceptance Test Plan
Acceptance Test Report
Analysis Model
Change Order
Concept Of Operations
Configuration Management Plan
Conversion Plan
Cost Benefit Analysis
Customer Approval
Customer Satisfaction Report
Data Model Item
Data Model Specification
Deployment Plan
Design Model
Development Contingency Plan
Documentation Plan
Integration Plan
Interface Control Document
IT Nomination Packet
Maintenance Plan
Master Test Plan
Measurement Plan
Phase Review
Programming Standards
Project Boundary Document
Project Cost Document
Project Performance Report
Project Plan
Project Proposal
Project Status Report
Project Tasklist
Project Timeline
Quality Assurance Plan
Requirements Document
Risk Assessment
Risk Management Plan
Role Statement
Security Plan
Storyboard
System Administration Plan
System Test Plan
Training Courseware
Training Plan
Training Survey
Unit Test
Unit Test Plan
Use Case Business Model
Use Case Model
User Interface Mockup
User Response Survey
User Support Plan
Workscope

Appendix E – SDLC Forms

The following are the Word document versions of the forms listed in Appendix D, and referenced in Appendix C. They were generated from XML descriptions of the forms. XML was used to describe the forms because it is easily understood and it is also, with the use of XSLT processing, easily converted to any desirable form, such as web pages or Word documents.

State of Montana
Department of Transportation
Acceptance Test Use Case

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Use Case

C.Test Modes

D.Failure Modes

E.Threat Modes

State of Montana
Department of Transportation
Acceptance Test Plan

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.System Description

Attach any relevant documents.

C.Test Environment

Attach any relevant documents.

D.Test Procedures

E.Acceptance Test Use Cases

Attach all relevant Acceptance Test Use Cases.

State of Montana
Department of Transportation
Acceptance Test Report

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Test Analysis

Attach any relevant documentation

C.Engineering Recommendation

Check one.

- ☐ Full acceptance
- ☐ Full acceptance with conditions
- ☐ Partial acceptance
- ☐ Rejected

D.Conditions

a.Expected Feature Conditions

b.New Feature Conditions

c.Time Conditions

d.Other Conditions

E.Change Requests

Attach change requests as required.

State of Montana
Department of Transportation
Analysis Model

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Use Case Reduction

C.Collaboration Analysis

Attach UML diagrams.

D.State Diagrams

Attach State Diagrams.

E.Classes

Attach Package Diagrams.

F.Verification

Attach relevant documentation.

G.Packages

Attach Package Diagrams.

State of Montana
Department of Transportation
Change Order

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Change Requirements

Change Order Number:	Version:
Responsible Team:	Team Member:
Criticality:	Due Date:
Status:	Date Changed:

C.Change Description

D.Test Procedures

State of Montana
Department of Transportation
Concept Of Operations

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.System Description

Attach any relevant documents.

a.Goals and Objectives

b.Rationale

C.Automation

Attach any relevant documents.

D.Functional Requirements

Attach any relevant documents.

E.Non-functional Requirements

Attach any relevant documents.

F.Operational Requirements

a.Deployment and Support

b.Configuration and Implementation

c.System Environment

G.User Classes and Operational Modes

a.Functional Usage

b.Sample Scenarios

H.Organizational Considerations

a.Impacts

b.Potential Risks

I.Assumptions

Attach any relevant documents.

J.Constraints

K.Preferences

Attach all relevant documentation.

L.Preferences

Attach all relevant documentation.

State of Montana
Department of Transportation
Configuration Management Plan

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Scope

Attach any relevant documents.

C.Management

Attach any relevant documents.

a.Terms

b.Documents

c.Organization

d.Responsibilities

D.Description

Attach any relevant documents.

a.Identification

b.Tools

c.Files

E.Configuration Control

F.Accounting

G.Integration

State of Montana
Department of Transportation
Conversion Plan

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Purpose and Scope

Attach any relevant documents.

C.Points of Contact

Attach any relevant documents.

D.System Overview

Attach any relevant documents.

E.Conversion Overview

F.Conversion Strategy

G.Conversion Tasks

Attach any relevant documents.

a.Conversion Planning

b.Pre-conversion Tasks

c.Conversion Tasks

d.Conversion Schedule

State of Montana
Department of Transportation
Cost Benefit Analysis

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Business Problem

a.Current Process

b.Problems

c.Impact

d.Expected Completion Date

e.Current Technology Environment

C.Alternatives

D.Costs and Benefits

E.Cost of Ownership

F.Recommendation

Attach relevant documentation.

G.Proposed Environment

Attach relevant documentation.

H.Proposed Schedule

Attach relevant documentation.

I.Risk Assessment

State of Montana
Department of Transportation
Customer Approval

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Approval

Accepted By: Date:

C.Comments

Attach any relevant documents.

State of Montana
Department of Transportation
Customer Satisfaction Report

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Overall Satisfaction

Please be specific.

C.Specific Areas of Dissatisfaction

Be specific.

D.Areas of Satisfaction

Please be specific.

E.Recommendations

Please be specific.

State of Montana
Department of Transportation
Data Model Item

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Data Name

C.Data Object

D.Data Types

E.Methods

F.Security

G.Compliance

H.Relationships

I.Special Needs

State of Montana
Department of Transportation
Data Model Specification

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Overview

C.Data Descriptions

a.Data Object

b.Data Object

D.Specific Areas of Dissatisfaction

Be specific.

E.Areas of Satisfaction

Please be specific.

F.Recommendations

Please be specific.

State of Montana
Department of Transportation
Deployment Plan

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Target System Description

C.Deployment Unit

D.Support

E.Training

F.Task List

G.Schedule

State of Montana
Department of Transportation
Design Model

A.General Information

Project Name:	**Date:**
Controlling Agency:	**Modification Date:**
Prepared By:	**Authorized By:**

B.Naming Convention

C.Design Packages

D.Class Extension

Attach UML diagrams.

E.Interfaces

Attach Interface Diagrams.

State of Montana
Department of Transportation
Development Contingency Plan

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Overview

C.Risk Response

D.Actions

State of Montana
Department of Transportation
Documentation Plan

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Overview

C.Documentation Manifest

D.Publication Standards

Attach any relevant documents.

E.Procedures

F.Sign-off

G.Schedule

State of Montana
Department of Transportation
Integration Plan

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Releases

C.Environment

D.Configuration

E.Process

F.Integration Test Plan

G.Scheduling

H.Error Handling

State of Montana
Department of Transportation
Interface Control Document

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Overview

C.Interactions

a.Functional

b.Data Formats

c.Data Communications

d.Protocols

e.Operation

State of Montana
Department of Transportation
IT Nomination Packet

A.District or Division

Check one.

- ☐ Billings District
- ☐ Butte District
- ☐ Glendive District
- ☐ Great Falls District
- ☐ Missoula District
- ☐ Aeronautics Division
- ☐ Administration Division
- ☐ Director's Office
- ☐ Engineering Division
- ☐ Human Resources Division
- ☐ Information Services Division
- ☐ Internal Audit
- ☐ Legal Services
- ☐ Maintenance Division
- ☐ Motor Carrier Services Division
- ☐ Rail, Transit and Planning Division

B.Year of Project Initiation

C.Proposed Completion Date*

D.Project Name

E.Project Type

Check one.

- ☐ New Customized Application
- ☐ Commercial Off-the-Shelf
- ☐ Major Enhancement to an Existing Application
- ☐ Non-peripheral IT
- ☐ Hardware
- ☐ Other _____

F.Project Description

G.Proposed Funding Source

Check one.

- ☐ ISD Funded
- ☐ Division or District Funds
- ☐ EPP Request
- ☐ Grant
- ☐ Federal or State Earmark
- ☐ Other agency funds _____
- ☐ Other outside funds _____

H.Project Category

Check one.

- ☐ "Must-do" Project
- ☐ Competitive Project

I."Must-do" Justification*

J.Business Process Documented

Check one.

- ☐ Yes (attach business process documentation)
- ☐ No

K.Business Process Owner

L.Project Priority

M.MDT Strategic Goal Supported

N.Department Criticality Criteria Justification

O.External Customer Criteria Justification

P.Internal Customer Criteria Justification

State of Montana
Department of Transportation
Maintenance Plan

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Overview

C.Maintenance Lifecycle

D.Environment

E.Roles and Responsibilities

F.Procedures

State of Montana
Department of Transportation
Master Test Plan

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Project Objectives

C.Unit Testing

D.Integration Testing

E.System Testing

F.Other Tests

State of Montana
Department of Transportation
Measurement Plan

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Objectives

C.Metrics

D.Data

E.Analysis

F.Reporting

G.Auditing

State of Montana
Department of Transportation
Phase Review

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Scope

C.Points of Contact

D.Overview

Attach relevant documentation.

E.Action Items

State of Montana
Department of Transportation
Programming Standards

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Scope

C.Organization

D.Layout

E.Naming Conventions

F.Structure

G.Replacements

State of Montana
Department of Transportation
Project Boundary Document

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Mission

C.Critical Success Factors

D.Requirements

a.Existing Methods and Procedures

b.Functional Requirements

c.Data Requirements

d.Technical Framework

e.Interface Requirements

E.Impacts

F.Technological Assumptions and Constraints

Attach relevant documentation.

G.Management Assumptions and Constraints

Attach relevant documentation.

State of Montana
Department of Transportation
Project Cost Document

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Data

C.Action Items

State of Montana
Department of Transportation
Project Performance Report

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Project Objectives

C.Results

D.Project Operational Objectives

E.Operational Results

Attach relevant documentation.

F.Successes

Attach relevant documentation.

G.Failures

Attach relevant documentation.

H.Recommendations

Attach relevant documentation.

State of Montana
Department of Transportation
Project Plan

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Objectives

C.Development Strategy

D.Management Roles

E.Project Team

Attach relevant documentation.

F.Tasks

Attach relevant documentation.

G.Schedule

Attach relevant documentation.

State of Montana
Department of Transportation
Project Proposal

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Project Description

C.Attachments

State of Montana
Department of Transportation
Project Status Report

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Significant Accomplishments

C.Schedule Variations

D.Resources Used

Attach relevant documentation.

E.Upcoming Issues

Attach relevant documentation.

State of Montana
Department of Transportation
Project Tasklist

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Tasks

C.Resources

D.Schedule

State of Montana
Department of Transportation
Project Timeline

A. General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B. Timeline

State of Montana
Department of Transportation
Quality Assurance Plan

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Organization

C.Software Development

Attach relevant documentation.

D.Documentation

Attach relevant documentation.

E.Deployment

Attach relevant documentation.

F.Training

Attach relevant documentation.

State of Montana
Department of Transportation
Requirements Document

A.General Information

Project Name:	**Date:**
Controlling Agency:	**Modification Date:**
Prepared By:	**Authorized By:**

B.Concept of Operations Document

Attach the Concept of Operations Document

C.Business Process Use Cases

Attach all relevant documentation.

D.Client Response

Attach all relevant documentation.

E.System Specification

Attach all relevant documentation.

F.Client Response

Attach all relevant documentation.

G.Design Model

Attach all relevant documentation.

H.Client Response

Attach all relevant documentation.

I.Interim Testing

Attach all relevant documentation.

State of Montana
Department of Transportation
Risk Assessment

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Scope

C.Requirements Development

Attach relevant documentation.

D.Design

Attach relevant documentation.

E.Software Development

Attach relevant documentation.

F.Deployment

Attach relevant documentation.

G.Mitigation

Attach relevant documentation.

H.Assessment

State of Montana
Department of Transportation
Risk Management Plan

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Risk Reduction Analysis

C.Action Plans

Attach relevant documentation.

State of Montana
Department of Transportation
Role Statement

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Role Statement

C.Role Description

Attach relevant documentation.

D.Deliverables

Attach relevant documentation.

E.Performance Review

Attach relevant documentation.

State of Montana
Department of Transportation
Security Plan

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Overview

C.Applicable Laws

Attach relevant documentation.

D.Risk Assessment

Attach relevant documentation.

E.Current Controls

Attach relevant documentation.

F.Proposed Controls

Attach relevant documentation.

G.Tools

Attach relevant documentation.

H.Training

Attach relevant documentation.

State of Montana
Department of Transportation
Storyboard

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Use Case Identifier

C.Description

Describe the process represented by this storyboard.

D.Storyboard Details

Attach the storyboard.

E.Feedback

State of Montana
Department of Transportation
System Administration Plan

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.System Organization

C.Inventory

D.Personnel

E.Tasks

F.Operations

Attach relevant documentation.

G.Controls

State of Montana
Department of Transportation
System Test Plan

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Project Objectives

C.System Objectives

D.Test Organization

E.References

F.Features to Test

Attach relevant documentation.

G.Features Not to Test

H.Testing Approach

I.Test Data

J.Controls

K.Test Documents

L.Test Environment

M.Staffing and Schedule

State of Montana
Department of Transportation
Training Courseware

A. General Information

Project Name:	**Date:**
Controlling Agency:	**Modification Date:**
Prepared By:	**Authorized By:**

B. Training Courseware

Attach training courseware or list references.

State of Montana
Department of Transportation
Training Plan

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Objectives

C.Curricula

Attach relevant documentation.

D.Skill Level

E.Personnel

Attach relevant documentation.

F.Evaluation

G.Scheduling

H.Roles

I.Development Cycle

J.Facilities

K.Performance

State of Montana
Department of Transportation
Training Survey

A.General Information

Project Name:	**Date:**
Controlling Agency:	**Modification Date:**
Prepared By:	**Authorized By:**

B.Content

Attach the survey form

State of Montana
Department of Transportation
Unit Test

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Module

C.Roles

D.Tests

Attach relevant documentation.

Input Data	Expected Output	Actual Output	Failed/Passed

E.Coments

Attach relevant documentation.

State of Montana
Department of Transportation
Unit Test Plan

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Overview

C.Modules To Test

Attach relevant documentation.

D.Modules Not To Test

Attach relevant documentation.

E.Procedures

Attach relevant documentation.

F.Controls

State of Montana
Department of Transportation
Use Case Business Model

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Business Objectives

C.Business Use Cases

Attach relevant documentation.

D.Actors

Attach relevant documentation.

E.Use Cases

Attach Use Case Diagrams.

F.Events

G.Schedule

State of Montana
Department of Transportation
Use Case Model

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Project Objectives

C.Functional Analysis

Attach relevant documentation.

D.Actors

Attach relevant documentation.

E.Events

F.Use Cases

Attach Use Case Diagrams.

G.Tasks and Schedule

Attach Use Case Diagrams.

State of Montana
Department of Transportation
User Interface Mockup

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Identifier

C.Interface Description

D.Mockup Details

E.Mockup Documentation

F.Feedback

State of Montana
Department of Transportation
User Response Survey

A. General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B. Content

Attach the survey form.

State of Montana
Department of Transportation
User Support Plan

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Objectives

C.Organization

D.Metrics

State of Montana
Department of Transportation
Workscope

A.General Information

Project Name:	Date:
Controlling Agency:	Modification Date:
Prepared By:	Authorized By:

B.Deliverables

C.Tasks

D.Schedule

E.Resources

F.Risks

This document was published electronically at an estimated cost of $0.00 each, for a total cost of $0.00. This includes $0.00 for postage and $0.00 for printing.