Fachhochschule Südwestfalen
Hochschule für Technik und Wirtschaft
University of Applied Sciences

Managing Small Software Projects - An Integrated Guide Based on PMBOK, RUP, and CMMI

César Cid Contreras M.Sc.
Prof. Dr. Henrik Janzen

Published at the South Westphalia University of Applied Sciences, Campus Soest, in August 2006.

ISBN-10: 3-939410-01-2

ISBN-13: 978-3-939410-01-0

Abstract

Software projects are generally complex and the environment in which they are developed is dynamic. The dynamics of this environment are based i.e. on the business conditions and the technological changes that appear during the project. These conditions lead to the result, that the software industry has continuously cost overruns, late deliveries, poor reliability, and user dissatisfaction. One cause of these deviations lays on the poor or improper project management ascription that allows a project to be out of control quickly. This work develops a guide, which is based on the body of knowledge of project management, the Rational Unified Process and the capability of assessment standard CMMI-SW (Staged) Level 2, to exercise executive, administrative, and supervisory direction of small software development projects. The guide provides orientation to project managers of small projects for the application of PMBOK in the development of software within the framework of the standard CMMI level 2 and the Rational Unified Process. It is concluded that the guide provides a path and foundation to new managers, new project leaders, and experienced ones as well to manage small software development projects effective. The result of using the guide should be improvements in cost, productivity, defects, schedule, and business value.

Keywords

Project Management, PMBOK, Rational Unified Process, RUP, Capability Maturity Model Integration, CMMI, software development.

1. Background

Many of the software development efforts are considered as projects. Software projects are generally complex and the environment in which they are developed is dynamic. The dynamics of this environment are based on the business conditions and the technological changes that appear during the project. Users are not sure of their needs and change their requirements several times during the project. These conditions lead as a result, that the software industry has continuously cost overruns, late deliveries, poor reliability, and user dissatisfaction [1].

The managing of projects is difficult and software development projects are not so far from this. Some difficulties have their origin in the own characteristics of the product, others have a relationship with management [1]. Although every software project has to overcome technical difficulties, these are not the main reason of the failure of projects [2].

The real important project shortfall lays on the poor or improper project management ascription that allows a project to be out of control quickly. If the project-related goals are not fulfilled, the project will fail and the frustrated members of the project will be addressed to another task [2].

Project managers play a crucial role in software projects and can be a major source of errors that lead to failure [3]. The project manager is responsible for project planning and estimation, control, organization, contract management, quality management, risk management, communications, and human resource management. Bad decisions by project managers are probably the single greatest cause of software failures today [3].

Because of the fact that software development can be considered as a project, essential definitions of project, management, and project management are required. For this aim, Project Management Institute (PMI) *Project Management Body of Knowledge* (*PMBOK*® 2004) is selected. The *PMBOK® Guide* is a standard that describes best practices for what should be done to manage a project. It covers nine areas that contain relevant knowledge. The practices and knowledge described on it are applicable to most projects most of the time, and that there is also a widespread consensus about their value and usefulness.

Careless software development practices are rich a source of failure too, and they can cause errors at any stage of a software project [4]. To help organizations assess their software-development practices, the U.S. Software Engineering Institute (SEI) created the Capability Maturity Model (CMM). It rates a company’s practices against five levels of increasing maturity. The SEI’s CMM has gained popularity in recent years for assessing and improving software processes. CMM may be defined as a degree to which an organization is in fact using an orderly software development process. CMM-Integration (CMMI) supersedes is superseding CMM and aims for a broader assessment of an organization’s ability to create software-intensive systems. Adoption of CMMI allows expanding the scope of and visibility into the software life cycle and activities to ensure that the software product meets customer expectations [4].

We take the Rational Unified Process as our product life cycle- software development life cycle actually-. The RUP is considered as a software development approach, well-defined and well-structured software engineering process, and a process product. As a software development approach, RUP is an iterative software life cycle, architecture-centric, and use-case-driven [5]. We are interested in these features because it promotes delivering of executable software at early stages of the software life cycle. Production of executable software is a good parameter to keep a real tracking of the status of the project [5]. The phases of the RUP are Inception Phase, Elaboration Phase, Construction Phase, and Transition Phase.

2. Guide for managing small software projects

The guide is integrated by the use of the better practices provided by the PMBOK® Guide and the key processes established by the CMMI-SW (Staged) Level 2, to exercise executive, administrative, and supervisory direction of small software development projects. The integration will be done within the Rational Unified Process (RUP) phases. The RUP is a software development approach that helps us to define our product life cycle in which the guide is addressed (Figure 2.1).

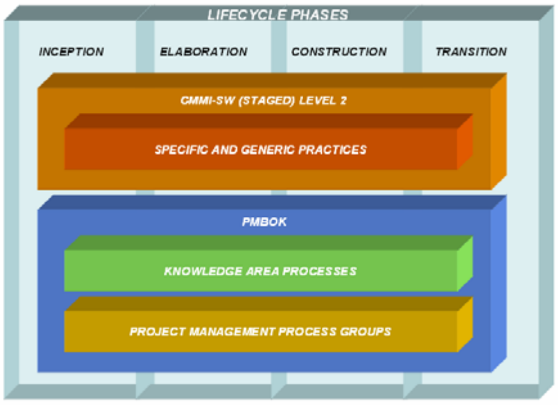

Figure 2.1 Implementation of the PMBOK and the CMMI-SW (Staged) Level 2 within the phases of the software development lifecycle proposed by the RUP.

2.1 Inception Phase

Inception is the first of the four phases of the RUP. It is about understanding the project scope and objectives and getting enough information to confirm that the project can be addressed or not. The objectives of the Inception phase [5] are taken to develop our first steps to achieve the first phase of the RUP. The objectives of the Inception phase are required to be done as parallel activities.

2.1.1 Objectives of the Inception Phase

Understand what to build.

- Identify key stakeholders. Identification of individuals and organizations that are actively involved in the project, or who interest may be affected as a result of project execution or project completion [6]. The contact information of the stakeholders is as important as the identification of them. Their contact information should be obtained to keep always channels of communication with them.

- Organize teams. Organize the project around cross-functional teams containing analysts, developers, testers, [5] and, if possible, users. If the project is bigger, we organize the projects around the architecture. The architecture team decides on the subsystems and the interfaces between them. Teams communicate with other teams primarily through the architecture and the architecture team [5] (see Figure 2.2).

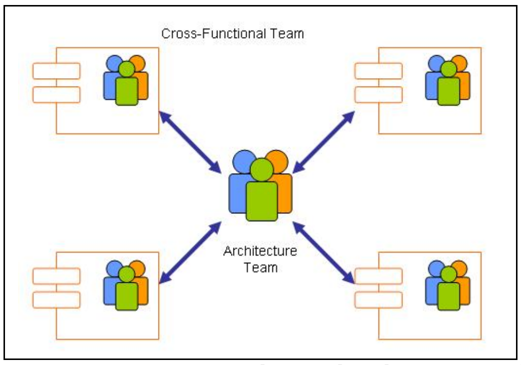

Figure 2.2 Teams organized around architecture. Is the project is too big to have everyone on one team? We may organize teams around architecture in “team of teams”. An architecture team owns the subsystems and their interfaces, and a cross-functional team is responsible for each of the subsystems [5].

- Write a vision document. The Vision document creates the foundation for common understanding of the motivation for building the system, as well as a high-level definition of the system to be built [5]. The vision should be public shared, and constantly reviewed with the stakeholders. Points described below will be also part of the Business Case document. The Business Case describes the economic value of the product, expressing it in quantitative terms such as, for example return on investment (ROI). Points to be considered in the vision document are the following [5]:

        - Introduction.
        - Business objective.
        - Current situation and problem/opportunity statement.
        - Critical assumptions and constraints, e.g. nonfunctional requirements.
        - Preliminary project requirements (use case model).
        - Glossary.

Identify key system functionality.

- Identify critical use cases. Some criteria for the key-use cases are [5]:
    - The use case is the core of the application.
    - The use case exercises key interfaces of the system.
    - The use case captures the essence of the system, and delivering the application without it would be fruitless.
    - The use case covers an area of the architecture that is not covered by any other critical use case.

Determine at least one possible solution.

- Some points to consider for determining a potential architecture are [5]:
    - Desired functionality (first version, as well as future versions of the application).
    - Compatibility with other applications.
    - Requirements on operations.
    - Maintenance.

- Look for options. Some considerations should be taken into account to facilitate our decision making [5]:
    - Are there other similar systems that were built within our organization or outside?
        - What technology and architecture were used on them?
        - What was the cost?
    - Is the current technology still adequate?
    - What technologies would be used within the new system?
    - Is it necessary to acquire new technologies?
        - What are their risks and costs?
    - What are the components needed for the system?
        - Can these components be purchase?
        - Can they be reused from another in-house project?
        - What are their risks and costs?

- Write the second part of the business case. On this second part of the Business Case, we can describe briefly the options for addressing the challenge and what from our point of view is the best option [5].

- Implement some key elements of the architecture (if applicable). The implementation in software of key elements will help us to identify risks and options for the architecture that should be developed [5].

Understand the costs, schedule, and risks associated with the project.

- Write the third part of the business case. Points to be considered in the vision document are the following [5]:
    - Budget estimate and financial analysis.
    - Schedule estimate.
    - Potential risks.
    - Exhibits.

- Check with stakeholders. The documents that should be checked together with the stakeholders and signed by them also are [5]:
    - Business Case.
    - Vision.
    - Project Charter.
    - Software Development Plan.

Decide what process to follow and what tools to use.

- Process. We decide how we are going to develop software, i.e. the process to follow. This process should be shared among all the team members [5].

- Tools. Once we have decided on a process, we can choose what tools to use. In some cases, the tool environment may already be decided. If not, then we need to choose which Integrated Development Environment (IDE), requirements management tool, visual modeling tool, configuration and change management tool , and so on to use [5].

- Artifacts to produce. Artifacts are the tangible project elements (things the project produces or uses while working toward the final product) [5].

- Templates. The project manager and the architect should settle what templates to use, and how to document the information [5].

2.1.2 Suggested iterations and outputs

Iterations.
We suggest two iterations for this phase of the project. In the first iteration, analysts and stakeholders should write a draft of the Vision document, use cases model, and a glossary of terms used in the project. It should be also considered the development of a conceptual prototype to help clarifying and agreeing in use cases with stakeholders. The project manager and the architect should think about the adequate process and tools which are the most suitable for the project.

During the second iteration, the team refines the Vision document with the feedback given by the stakeholders, the most critical use cases are described in detail and updates to the use case model are done. Based on the detailed description of the critical use cases, the implementation of a functional prototype should be achieved, thus it will help to determine what technology and tools to use within the project.

Outputs.
The outputs of this phase are as follows:

- Contact information of the key stakeholders.
- Vision document.
- Business Case.
- Project Charter.
- Software Development Plan.

Regarding the Project Charter, this document should be preferably one or two pages long, and it may refer to other documents, such as business case, as needed. The signatures of key stakeholders and their individual comments are the most important parts [6].

Points the Project Charter may contain are the following [6]:

- Project Title.
- Project start date.
- Projected finish date.
- Budget information.
- Project manager name.
- Project objectives
- Approach to fulfill the requirements.
- Roles and responsibilities of key stakeholders.
- Comments of the stakeholders.

2.1.3 Planning for the Inception Phase

Two kinds of plans are developed [5]:

- Project Plan. This is a coarse-grained plan, which focuses on phases and iterations, their objectives, and the overall staffing level.
- Iteration Plans. These are a series of fine-grained plans, one per iteration, which bring activities and individual resources into perspective.

2.1.3.1 Project plan.

The project manager may collaborate closely with the development team or the architect to determine an initial estimate of the overall size of the project. Consideration of historical data is also helpful when the current project is compared with previous similar projects.

Determine dates of major milestones.
Milestones considered for every phase in the RUP are the following [5]:

- Life Cycle Objective (LCO) Milestone. End of Inception, project well scoped and funded.
- Life Cycle Architecture (LCA) Milestone. End of Elaboration, architecture complete, requirements baseline set.

- Initial Operational Capability (IOC) Milestone. End of Construction, first beta release.
- Product Release (PR) Milestone. End of Transition and of the development cycle.

Determine staffing profile.

Staffing is the allocation of the right level of resources to the project alongside the lifecycle. Figure 2.3 shows a typical staffing profile.

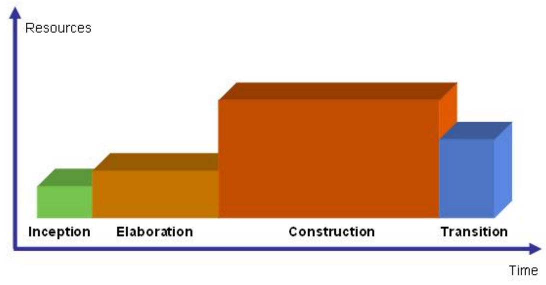

Figure 2.3 Typical Resource Profile for a Development Cycle. The resources used within each phase vary greatly from project to project. This graph provides us with a starting point for a discussion around resource needs [5].

Determine iterations.

Related to the number of iterations is the issue of the length of an iteration (see Table 2.1). As first approximation, obtain the iteration length by dividing the length of the phase by the number of iterations. If the duration obtained is not quite right, revisit the process [5].

Project Size (People)	Project Length (Months)	Iteration Length (Weeks)	Number of Iterations Per Phase			
			Inception	Elaboration	Construction	Transition
3	4	2-3	1	1	3	1
10	8	4	1	2	3	2

Table 2.1 Degrees of iteration in different projects. This table can be used as a starting point when deciding how many iterations to have (took it partially from [5] for small projects).

2.1.3.2 Iteration plan.

As the iteration plan focuses on one only iteration, it has a time span small enough that it provides team members with a plan that includes the right level of granularity on tasks and successful allocation to various team members [5]. A project usually has two iterations plans "active" at any time [5]:

- Current Iteration Plan: it is for the current iteration, which is used to track progress.
- Next Iteration Plan: it is for the upcoming iteration, which is built toward the second half of the current iteration and is ready at the end of the current iteration.

We can read on Schwalbe [6] the application of the PMBOK Guide to a software development project. In this study, we apply the PMBOK Guide to every iteration plan within every phase of the RUP software life cycle. We follow the basic structure provided by Schawalbe [6] but taking into account the objectives and outputs required by the RUP software life cycle and the CMMI (staged) Level 2.

First iteration plan.

For the first phase of the development cycle, we suggest two iterations; the plan for the first iteration is described as follows:

1. Current Iteration Plan

1.0 Initiating.
1.1 Identify key stakeholders.
1.2 Prepare Vision document.

2.0 Planning.
2.1 Hold project kick-off meeting.
2.2 Prepare WBS.
2.3 Identify, discuss, and prioritize risks.
2.4 Prepare schedule and cost baseline for.

- Determine task resources.
- Determine task durations.
- Determine task dependencies.
- Create draft Gantt chart.
- Review Gantt with stakeholders, obtain commitment, and finalize Gantt chart.

3.0 Executing.
3.1 Create draft Use Case Model.
3.2 Create Glossary.
3.3 Develop Conceptual Prototype (if applicable).
3.4 Establish Configuration and Control Management.

4.0 Controlling.
4.1 Status reports.
4.2 Review of conceptual prototype (if applicable).

5.0 Closing
5.1 First draft Vision
5.2 First draft Use Case Model.
5.3 First draft Glossary Model.
5.4 Conceptual prototype (if applicable).
5.5 Possible options of process and tools for fulfill the project.
5.6 Configuration and Control Management.
5.7 Lessons learned document.

Second iteration plan.
The plan for the second iteration is described as follows:

2. Next Iteration Plan

1.0 Initiating.
1.1 Refine Vision document.

2.0 Planning.
2.1 Prepare WBS.
2.2 Identify, discuss, and prioritize risks.
2.3 Prepare schedule and cost baseline for.

- Determine task resources.
- Determine task durations.
- Determine task dependencies.
- Create draft Gantt chart.
- Review Gantt with stakeholders, obtain commitment, and finalize Gantt chart.

3.0 Executing.
3.1 Identify Key System Functionality.
3.2 Detail Critical Use Cases.
3.3 Determine one possible solution (write 2nd part of the business case).
3.4 Develop functional prototype (if applicable).
3.5 Determine costs, schedule, and risks for the next phases of the development process (write 3rd part of the business case).
3.6.Check with stakeholders:

- Business case
- Vision
- Project Plan
- Project Charter

3.7 Determine process to follow and tools to use.

4.0 Controlling.

2.2 Elaboration Phase

Elaboration is the second phase of the RUP. It addresses major risks, builds an early skeleton architecture of the system, and refines and evolves the project plans that were produced in Inception. Risks associated with requirements, the architecture, costs ad schedule, process and tool environment are addressed in this phase [5].

2.2.1 Objectives of the Elaboration Phase

Get more detailed understanding of the requirements.
By the end of the Inception phase, we should have detailed the critical use cases in our use case model. These architecture significant use cases should be the 20% of the total use cases. By the end of the Elaboration phase, we should have a complete description of the majority of the use cases. It is important to have the description of the use cases but also a prototype. The user will interact with the prototype as we test each use case with him/her. The interaction will clarify the user what information is displayed and entered. Feedback from the user is valuable through the entire project but more important in this phase [5].

Design, implement, validate, and baseline the architecture.

- Architecture: defining subsystems, key components, and their interfaces. Rather than inventing a new architecture, we should first envisage whether there is an architectural framework available, commercial architecture or a similar architecture that we developed before from a previous work. If there is not such architecture, then we have to identify the major building blocks, that is, the subsystems and major components. For each identified subsystem or component, we should describe the key capabilities they need to offer, namely, their interfaces toward the rest of the system. In parallel with identifying key components and subsystems, we need to survey available assets inside and outside the company [5].

- Use architecturally significant use cases to drive the architecture. The critical use cases identified in the Inception phase are likely to be significant in driving the architecture. Other aspect to take into account in driving the architecture is the nonfunctional requirements. The nonfunctional requirements are technical challenges to the infrastructure part of the architecture, for example response time, load, and error recovery [5]. Finally, we should identify some use cases that, although not critical nor technically challenging, address some parts of the system not yet covered, so we can have a complete control of the entire architecture an the end of Elaboration. We must ensure that the architecture will us to deliver all the architecturally significant use cases by designing, implementing, and testing as many of these use cases as necessary to mitigate the risks associated with them [5].

- Design the database. If our solution includes a database where data is retrieved and stored, we should start the design of it [5].

- Outline concurrency, processes, threads, and physical distribution. The main goal of this objective is to describe the runtime architecture in terms of concurrency, processes, threads, interprocess communication, and so on [5].

- Identify architectural mechanisms. Architectural mechanisms represent common concrete solutions to frequently encountered problems [5]. The most common and difficult problems can be solve once by designing, implementing, testing, and documenting architectural mechanisms. Then, all team members can take advantages of these ready-made solutions whenever they need them.

- Implement critical scenarios. We can design a little, implement what we design, detect deficiencies, and then improve the design. We should also develop test documentation to make sure our implementations perform according to specifications [5].

- Integrate components. Integration and testing are common tasks when doing iterative development. As we do analysis and design, we should determine the order and the components we want to integrate, so we can verify our design and implement the functionality necessary to integrate and compile the evolving system for testing [5].

- Test critical scenarios. By testing we want to verify that [5]:

    - Critical scenarios have been properly implemented and offer the expected functionality.
    - The architecture provides sufficient performance.
    - The architecture can support necessary load.
    - Interfaces with external systems work as expected.
    - Any other requirements in the supplementary specification (non functional requirements) that are not captured above are tested.

Mitigate essential risks, and produce accurate schedule and costs estimates.
Toward the end of Elaboration, we have the following information [5]:
- Detailed requirements.
- Implementation of a skeleton structure (executable architecture).

- Mitigation of the vast majority of risks.
- Understanding of how effectively we are working with the people, the tools, and the technology.

This information will provide us more accurate information allowing us to update the Vision document, the project plan and cost estimate.

Refine process and tools, and put the development environment in place.
During Inception, we have defined what process to follow and tools to use and did necessary customization. In Elaboration, we update the process and fine-tune our tool implementation according to the experience that we have gained during these two phases. We also should outline what artifacts should be produced, what templates to use, and how to document information [5].

2.2.2 Suggested iterations and outputs

Iterations.
It is suggested two iterations for this phase of the project. In the first iteration, the following activities are considered [5]:

- Design, implement, and test a small number of critical scenarios.
- Identify, implement, and test a small set of architectural mechanisms.
- Do a preliminary logical database design.
- Detail flow of events of half of the use cases intended to detail in Elaboration.
- Test enough to validate that your architectural risks are mitigated.

In the second iteration the following activities are considered [5]:

- Fix whatever was not right in the first iteration.
- Design, implement, and test the remaining architecturally significant scenarios.
- Outline and implement concurrency, processes, threads, and physical distribution.
- Identify, implement, and test remaining architectural mechanisms.
- Design and implement a preliminary version of the database.
- Detail the second half of the use cases intended to detail in Elaboration.
- Test, validate, and refine the architecture to the point where it can be a baseline.

Outputs.
The outputs of this phase are as follows:

- Use case model update
  - Critical use cases detailed.
  - Complete description of use cases (about 80%).
  - Description of subsystems and interfaces.
- Database design.
- Architectural mechanisms document.
- Design of use cases document.
- Test plan.
- Development environment.
- Functional architecture.
- Project plan update.
- Vision update.

2.2.3 Planning for the Elaboration Phase

One of the priorities of the Elaboration phase is the mitigation of risks. Risks will determine which use cases and scenarios will be developed in each iteration [7]. For this Elaboration phase, we suggest 2 iterations. The first iteration will deliver a functional prototype of the architecture; during the second iteration, a baseline architecture should be completed.

2.2.3.1 Project plan

The project manager with the architect can update the estimates on the project plan based on the use case model and vision provided by the Inception phase. Consideration of historical data is also helpful when the current project is compared with previous similar projects.

Update dates of major milestones.

- Life Cycle Architecture (LCA) Milestone. Update end of Elaboration date.
- Initial Operational Capability (IOC) Milestone. Update end of Construction date.
- Product Release (PR) Milestone. Update end of Transition date.

Update staffing profile.
Update estimations about staffing for Elaboration phase and the following phases.

Update iterations.

Make adjustments either to the length of the phase or to the number of iterations.

2.2.3.2 Iteration plan.
We may have at least two iterations. The plan of both iterations is described below.

First iteration plan.
For the Elaboration phase, the plan for the first iteration is described as follows:

1. Current Iteration Plan

1.0 Initiating.
1.1 Refine Vision document.
1.2 Review use case model.

2.0 Planning.
2.1 Prepare WBS.
2.2 Identify, discuss, and prioritize risks.
2.3 Prepare schedule and cost baseline for.

- Determine task resources.
- Determine task durations.
- Determine task dependencies.
- Create draft Gantt chart.
- Review Gantt with stakeholders, obtain commitment, and finalize Gantt chart.

3.0 Executing.
3.1 Define subsystems, key components, and their interfaces.
3.2 Document subsystems, key components, and their interfaces (update use case model).

If a commercial framework is selected for the architecture, we should determine the acquisition type, select suppliers and establish supplier agreements.

3.3 Establish a deployment site and a deployment plan.
3.4 Select a small number of critical scenarios.
3.5 Design the small number of critical scenarios.
3.6 Design database.
3.7 Design a test plan for the small number of critical scenarios. Consider:

- Measures.
- Analysis of measures.
- Way of collecting measures.
- Report of measures.

3.8 Implement the small number of critical scenarios.
3.9 Test the small number of technical scenarios.
3.10 Create a 1st build.
3.11 Review 1st build with stakeholders.

- Compare 1st build with use case model.
- Create a verification report.

3.12 Fix what was wrong on the 1st build.
3.13 Identify, implement, and test architectural mechanisms.
3.14 Document architecture.
3.15 Document architectural mechanisms.
3.16 Update use case model.

Repeat from step 3.4 to 3.16 the times of builds we want to create.
(At least two builds are suggested per week).

3.16 Detail the events flow of the first half of the use cases intended to detail in Elaboration in order of decreasing risk.
3.17 Set development environment.

4.0 Controlling.
4.1 Status reports (weekly).
4.2 Review of the last build (functional architecture).

5.0 Closing
5.1 Use case model updated (half of the use cases intended to detail in Elaboration).
5.2 First designs of critical scenarios of architecturally use cases.
5.3 Architectural mechanisms document.
5.4 Architecture document.
5.5 Setting of development environment.
5.6 Process and tools refined.
5.7 Lessons learned document.

Second iteration plan.
The plan for the second iteration is described as follows:

2. Next Iteration Plan

1.0 Initiating.
1.1 Review of use case model.

2.0 Planning.
2.1 Prepare WBS.
2.2 Identify, discuss, and prioritize risk
2.3 Prepare schedule and cost baseline for.

- Determine task resources.
- Determine task durations.
- Determine task dependencies.
- Create draft Gantt chart.
- Review Gantt with stakeholders, obtain commitment, and finalize Gantt chart.

3.0 Executing.
3.1 Select the remaining critical scenarios.
3.2 Design the remaining critical scenarios.
3.3 Design database.
3.4 Implement database.
3.5 Design a test plan for the remaining critical scenarios.
3.6 Implement the remaining critical scenarios.
3.7 Test the remaining critical scenarios.
3.8 Create a 1st build.
3.9 Review 1st build with stakeholders.

- Compare 1st build with use case model.
- Create a verification report.

3.10 Fix what was wrong on the 1st build.
3.11 Identify, implement, and test architectural mechanisms.
3.12 Document architecture.
3.13 Document architectural mechanisms.
3.14 Update use case model.

Repeat from step 3.1 to 3.14 the times of builds we want to create.
(At least two builds are suggested per week).

3.15 Detail the events flow of the second half of the use cases intended to detail in Elaboration in order of decreasing risk.

4.0 Controlling.
4.1 Status reports (weekly).
4.2 Review of the last build (functional baseline architecture).

5.0 Closing
5.1 Use case model updated (complete description of the use cases, about 80% of the total use cases).
5.2 Documented designs of the critical scenarios.
5.3 Update of architectural mechanisms document.
5.4 Document of baseline architecture.
5.5 Process and tools baseline.
5.7 Lessons learned document.

2.3 Construction Phase

Construction focuses on detailed design, implementation, and testing to flesh out a complete system. During this phase, we focus on developing high-quality code cost-effectively. Keys of success in this phase are architectural integrity, parallel development, configuration and change management, and automated testing [5].

2.3.1 Objectives of the Construction Phase

Minimize development costs and achieve some degree of parallelism.

- Organize around architecture. A robust architecture divides the system responsibilities into well-defined subsystems. An architect or an architecture team worries about the architecture and how it all ties together, and individuals can focus on their assigned subsystem(s) [5].

- Configuration management. A configuration management system is used to track all versions of the many files being created and changed in the iterative development. With the help of a configuration management system we can to determine which version of these new or changed files goes into each build [5].

- Integration planning. Each iteration needs and integration build plan specifying which capabilities should be testable in each build and which components need to be integrated to produce required capabilities, such as use cases, part of use cases, or other testable functionality. The tests may include functional, load, stress, or other types of tests [5].

- Enforce the architecture. Developers should be trained on the architecture and architectural mechanisms available to prevent each developer from arbitrarily reinventing solutions for problems such as dealing with persistency or interprocess communication. This training process includes design reviews with architects and developers [5].

- Ensure continual progress. Some guidelines to consider for a continual progress are the following [5]:
    - Create one team with one mission. We should have cross functional teams, were each team member feels responsible for the application and for the team making progress.
    - Set clear, achievable goals for developers. Each developer should have a very clear picture of what to accomplish in a given iteration, if not within a portion of the iteration. The developers should agree that the expected deliverables are achievable.
    - Continually demonstrate and test code. Continual demonstration and testing of executable code is the only way to ensure progress.
    - Force continuous integration. Performing frequent builds ensures frequent integration testing, which provides feedback on the recent code that has been written since the last build.

Iteratively develop a complete product that is ready to transition to its use community.

- Describe the remaining use cases and other requirements. Nonessential use cases and those with no major architectural impact are generally skipped in Elaboration. Also, in some systems there are similar use cases, having the same general sort of functionality, but for different entities, or different actors, with different user interfaces. These types of use cases should be detailed in Construction, along with partially detailed use cases. Performance requirements or requirements related to application stability should be documented as well [5].

- Fill in the design. In earlier Construction iterations, we should design, implement and test only the most essential scenarios for the selected use cases. In later Construction iterations, we should focus on completeness until we eventually design, implement, and test all scenarios of the selected use cases [5].

- Design the database. In the Construction phase, additional columns may be added to tables, views may be created to optimize performance, but major restructuring of tables should not occur [5].

- Implement and unit-test code. Developers need to test their implementations continuously to verify that they behave as expected. Test drivers and test stubs may be designed and implemented to test component(s). Test drivers and test stubs emulate other components that will interact with the component(s); they also allow us to run a number of test scenarios [5].

- Do integration and system testing. When producing a build, components are integrated in the order specified in the integration build plan. To increase quality, continuously integrate and test our system [5].

- Early deployments and feedback loops. Performing frequent builds drives to continuous integration and verification that the code works. Integration and system testing also reveals many quality issues. It is crucial to get early feedback on whether the application is useful and provides desired behavior, by exposing it to actual users [5]. Some approaches for having feedback include [5]:
    - Bringing a few users to the development environment and demonstrating key capabilities.
    - Bringing a few users to the development environment and having them use the product for some time.
    - Installing the software at a test site and sitting with the users as they are using the software.
    - For hosted applications, providing some users with early access.

- Prepare for Beta deployment. A beta deployment is "prerelease" testing in which a sampling of the intended audience tries out the product. Beta deployment is done at the end of the Construction phase and is the primary focus of the Transition phase [5]. Beta testing has two purposes: First, it tests the application through a controlled actual implementation, and second, it provides a preview of the upcoming release. It is important that the product is complete, based on the scope management that has occur during iterations. The beta deployment should include installation instructions, user manuals, tutorials, and training material, so the testers can give also feedback on them [5].

- Prepare for final deployment. The final deployment should be done in Construction and sometimes earlier in Elaboration. Some activities regarding final deployment include [5]
  - Producing material for training users and maintainers to achieve user self-reliability later.
  - Preparing deployment site and converting operational databases.
  - Preparing for launch: packaging and production; preparing for rollout to marketing, distribution, and sales forces: preparing for field personnel training. These activities should take into account specially when developing a commercial product.

2.3.2 Suggested iterations and outputs

Iterations.
It is suggested three iterations for this phase of the project. In the first iteration, the following activities are considered [5]:

- Identify use cases that are
  - Most essential to customers.
  - Most technical risky.
- Classify use cases in order of decreasing risk.
- Identify components that need to collaborate together to achieve use case functionality.
- Implement only some scenarios within the most risky use cases.
- Update case model and architectural mechanisms.

In the second iteration the following activities are considered [5]:

- Keep implementing scenarios within the most risky use cases.
- Update use case model/architectural mechanisms.
- Start implementing al the scenarios of the use cases.

The third iteration includes the following activities [5]:

- Implement all the remaining scenarios of the use cases.
- Update use case model/architectural mechanisms.
- Write supporting documentation.

Outputs.
The outputs of this phase are as follows:

- Use case model update
  - All use cases detailed (100%).
  - Description of all subsystems and interfaces.
  - Description of subsystems and interfaces.
- Database design updated and completed.
- Architectural mechanisms document.
- Design of use cases document updated and completed.
- Test plan updated.
- Project plan update.
- Draft version of the supporting documentation.
- Beta release (functional system).

2.3.3 Planning for the Construction Phase

The activities of the Construction phase are detailed designing, implementation, and testing to develop a complete system [5]. For Construction, we suggest 3 iterations. In the first one, use cases that are most essential to customers will be implemented, as well those associated with most technical risk. This activity could be also extended to a part of the second iteration. We are going to implement all the use cases in the following iterations, having in mind a risk decreasing order.

2.3.3.1 Project plan

We have more information about the project from the previous phases and we have mitigated the main risks in previous phases, so we can have a better approximation of our estimates for the project plan and its iterations. The Construction phase is the phase of the lifecycle project that requires more staffing.

Update dates of major milestones.

- Initial Operational Capability (IOC) Milestone. Update end of Construction date.
- Product Release (PR) Milestone. Update end of Transition date.

Update staffing profile.

Update estimations about staffing for Construction phase and the following phases. A major participation of developers is expected in this phase.

Update iterations.

Make adjustments either to the length of the phase or to the number of iterations.

2.3.3.2 Iteration plan.

We may have at least three iterations.

First iteration plan.

For the Construction phase, the plan for the first iteration is described as follows:

1. Current Iteration Plan

1.0 Initiating.
1.1 Review Vision document.
1.2 Review use case model.

2.0 Planning.
2.1 Prepare WBS.
2.2 Identify, discuss, and prioritize risks.
2.3 Prepare schedule and cost baseline for.

- Determine task resources.
- Determine task durations.
- Determine task dependencies.
- Create draft Gantt chart.
- Review Gantt with stakeholders, obtain commitment, and finalize Gantt chart.

3.0 Executing.
3.1 Identify most essential to customer and/or most essential risky use cases.
3.2 Identify components that need to collaborate together to provide use case functionality.
3.3 Implement a couple of scenarios within the most essential to customer and/or most essential risky use cases.

- Design components.
- Design test plan for components. Consider:
  - Measures.
  - Analysis of measures.
  - Way of collecting measures.
  - Report of measures.
- Implement components.
- Test components.

3.4 Create a 1st build.
3.5 Review 1st build with stakeholders.

- Compare 1st build with use case model.
- Create a verification report.

3.6 Fix what was wrong on the first build.
3.7 Identify, implement, and test architectural mechanisms.
3.8 Document architectural mechanisms.
3.9 Update use case model.
3.10 Update database.

Repeat from step 3.3 to 3.9 the times of builds we want to create.
(At least two builds are suggested per week).

4.0 Controlling.
4.1 Status reports (weekly).
4.2 Review of the last build.
5.0 Closing
5.1 Use case model updated. Detailed description of essential to customer/risky use cases.
5.2 Design documents updated.
5.3 Architectural mechanisms document updated.
5.4 Database updated.
5.5 60% of the components implemented.
5.6 Lessons learned document.

Second iteration plan.

The plan for the second iteration is described as follows:

1. Next Iteration Plan

1.0 Initiating.
1.1 Review Vision document.
1.2 Review use case model.

2.0 Planning.
2.1 Prepare WBS.
2.2 Identify, discuss, and prioritize risks.
2.3 Prepare schedule and cost baseline for.

- Determine task resources.
- Determine task durations.
- Determine task dependencies.
- Create draft Gantt chart.
- Review Gantt with stakeholders, obtain commitment, and finalize Gantt chart.

3.0 Executing.
3.1 Implement all scenarios within the most essential to customer and/or most essential risky use cases.

- Design components.
- Design test plan for components. Consider:
  Measures.
  Analysis of measures.
  Way of collecting measures.
  Report of measures.
- Implement components.
- Test components.

3.2 Create a 1st build.
3.3 Review 1st build with stakeholders.

- Compare 1st build with use case model.
- Create a verification report.

3.4 Fix what was wrong on the first build.
3.5 Identify, implement, and test architectural mechanisms.
3.6 Document architectural mechanisms.
3.7 Update use case model.
3.8 Update database.

Repeat from step 3.1 to 3.8 the times of builds we want to create.
(At least two builds are suggested per week).

3.9 Detail the flow of events of the remaining use cases that were not covered in the Elaboration phase (20% of the total use cases).
3.10 Write supporting documentation.

4.0 Controlling.
4.1 Status reports (weekly).
4.2 Review of the last build.
5.0 Closing
5.1 Use case model updated. Detailed description of essential to customer/risky use cases and of the remaining use cases that were not covered in the Elaboration phase.
5.2 Design documents updated.
5.3 Architectural mechanisms document updated.
5.4 Database updated.
5.5 80% of the components implemented.
5.6 Draft of supporting documentation.
5.7 Lessons learned document.

Third iteration plan.
The plan for the third iteration is described as follows:

1. Iteration Plan after Next

1.0 Initiating.
1.1 Review Vision document.
1.2 Review use case model.

2.0 Planning.
2.1 Prepare WBS.
2.2 Identify, discuss, and prioritize risks.
2.3 Prepare schedule and cost baseline for.

- Determine task resources.
- Determine task durations.
- Determine task dependencies.
- Create draft Gantt chart.
- Review Gantt with stakeholders, obtain commitment, and finalize Gantt chart.

3.0 Executing.
3.1 Implement all scenarios within all use cases.

- Design components.
- Design test plan for components. Consider:
  - Measures.
  - Analysis of measures.
  - Way of collecting measures.
  - Report of measures.
- Implement components.
- Test components.

3.2 Create a 1st build.
3.3 Review 1st build with stakeholders.

- Compare 1st build with use case model.
- Create a verification report.

3.4 Fix what was wrong on the first build.
3.5 Identify, implement, and test architectural mechanisms.
3.6 Document architectural mechanisms.
3.7 Update use case model.
3.8 Update database.

Repeat from step 3.1 to 3.8 the times of builds we want to create.
(At least two builds are suggested per week).

3.9 Write supporting documentation.

4.0 Controlling.
4.1 Status reports (weekly).
4.2 Review of the last build.

5.0 Closing
5.1 Use case model detailed.
5.2 Design documents updated.
5.3 Architectural mechanisms document updated.
5.4 Database updated.
5.5 80% of the components implemented.
5.6 Draft of supporting documentation.
5.7 Lessons learned document.

2.4 Transition Phase

Transition starts with the beta deployment and concludes with final delivery of the solution to the customer or their support organization. This phase focuses on fixing remaining defects, training users, and, in many systems, converting data from older systems (or older versions of the same system) and running in a parallel testing mode for some period to ensure that the system is ready for final deployment [8].

2.4.1 Objectives of the Transition Phase

Beta test to validate that user expectations are met.

- Capturing, analyzing, and implementing change requests. Beta testing is done during Transition, which provides user feedback from the beta testers. Some activities to gather useful feedback are interviews, on-line queries, submitted change requests, among others. Then we should analyze the collected information, submit and review change requests with stakeholders to understand what changes are required before the final product release [5]. Change requests are mainly defects and beta test feedback. They are also the major planning input for continuing development. Mainly, the change requests are only for minor system small adjustments, such as fixing minor bugs, enhancing documentation or training material or tuning the performance [5]. Sometimes additional features must be added, that is, we have to work with requirements, analysis and design, implementation, and testing. This can be a sign of failure on earlier phases. In most cases we should refrain from adding new features and instead postpone them to a next development cycle, however, if the system requires these additional features for deployment, we should implement them [5]. Builds with incorrect file versions or missing files are common sources of defects at this stage. Good configuration management practices and tools reduce these types of errors [5]. During Transition, we should invest fair amount of time to improve documentation, online help, training material, user's guides, operational guides, and other supporting documentation. These elements should be tested by the beta testers in the target environment [5].

- Transition testing. In planning for Transition testing, we should provide effort and resources for the following [5]:
  - Continued test design and implementation to support ongoing development.
  - Regression testing. It will require variable effort and resources, depending on the chosen approach; for

example, retest everything or retest to an operational profile.
    - Acceptance testing, which may not require the development of new tests.

  As defects are fixed and beta feedback is incorporated, successive builds are tested using a standard test cycle [5]:
    - Validate build stability. A subset of test should be executed to validate that the build is stable enough to start detailed test and evaluation.
    - Test and evaluate. Implement, execute, and evaluate test.
    - Achieve test objectives. We should evaluate test results against testing objectives and perform additional testing as necessary.
      Improve test assets. We should improve test artifacts as needed to support the next cycle of testing.

- Patch releases and additional beta releases. A patch release is a special bug-fix release installed on top of the current baselined release. A patch is used, if serious defects that prevent effective beta testing are found [5].

- Metrics for understanding when transition will be complete. Defect metrics and test metrics, among other things, help determine when Transition will be complete [5].

    - Defect metrics: The important issue is to focus on the trend rater than the actual number of defects. By analyzing a defect trend, we can predict when a certain threshold value of open defects will be reached. Two aspects should be considered [5]
        - How many new defects are found each day.
        - How many defects are fixed each day.

    - Test metrics: We can predict how many new defects can be expected by determining how many defects are typically found per test case and multiplying that by the number of tests yet to be executed and analyzed [5].

Train users and maintainers to achieve user self-reliability.
Operational staff, all users, and maintenance teams should be trained during Transition. This training will also give feedback on training material, user documentation, and operational manuals. Training material and instructors training should be developed during Construction [5].

Prepare deployment site and convert operational databases.
One aspect to consider when deploying is the facilities where the final product will be installed. Some examples are new machines, power supply or back up power, network installation, and so on [5]. If the new system replaces an existing one, data needs to be transferred to the new system. Even when replacing a system, the new and old systems may need to run in parallel for some time to ensure correct performance of the new system. For complex deployments, these activities should be started in previous phases (Elaboration or Construction) [5].

Achieve stakeholder concurrence that deployment is complete.
Before deploying the software, there is one action to be considered: product acceptance testing. Acceptance testing verifies that the software is ready and can perform those functions and tasks it was built for [5].

Improve future project performance through lessons learned.
A post-mortem assessment is advisable at the end of each project. A post-mortem assessment consists in analyzing and documenting what worked well and what didn't. Based on the results, the development environment can be improved reflecting what was learned. Another point to consider is whether any work can be reused for other projects [5].

2.4.2 Suggested iterations and outputs

Iterations.
We suggest one iteration for this phase. The following activities are considered:

- Beta testing.
- Get feedback from beta testers.
- Capturing, analyzing, and implement change requests.
- Keep writing supporting documentation.
- Add new features (if applicable, but not suggested).
- Do deployment of the new system.
- Do Post-mortem assessment.

Outputs.
The outputs of this phase are as follows:
- Supporting documentation.
- Final release of software.
- Lessons learned document.

2.4.3 Planning for the Transition Phase

The focus of the Transition phase is to ensure that software is available for its end users. The Transition phase can span several iterations, and includes testing the product in preparation for release, and making minor adjustments based on user feedback [9]. For Transition, we suggest one iteration.

2.4.3.1 Project plan

At this phase, all major structural issues were solved in previous phases. The focus of Transition is on fine tuning the product, configuring, installing, and usability issues [5].

Update dates of major milestones.
- Product Release (PR) Milestone. Update end of Transition date.

Update staffing profile.
Update estimations about staffing for Transition phase and the following phases. A major participation of beta testers and end users is expected in this phase.

Update iterations.
Make adjustments either to the length of the phase or to the number of iterations.

2.4.3.2 Iteration plan.

One iteration is suggested. The plan of the iteration is described below.

First iteration plan.

For the Transition phase, the plan for the first iteration is described as follows:

1. Current Iteration Plan

1.0 Initiating.
1.1 Review Vision document.
1.2 Review use case model.

2.0 Planning.
2.1 Prepare WBS.
2.2 Identify, discuss, and prioritize risks.
2.3 Prepare schedule and cost baseline for.
- Determine task resources.
- Determine task durations.
- Determine task dependencies.
- Create draft Gantt chart.
- Review Gantt with stakeholders, obtain commitment, and finalize Gantt chart.

3.0 Executing.
3.1 Design a test plan for the beta release. Consider:
  Measures.
  Analysis of measures.
  Way of collecting measures.
  Metrics.
  Criteria for the evaluation.
  - Vision document.
  - Use cases model.

3.2 Achieve beta testing.
3.3 Get feedback from beta testing
  Capture
  - Report of measures.
  - Report of metrics.
  - Change requests.

  Analyze
    Report of measures.
    Report of metrics.
    Change requests.

3.4 Implement change request and bug-fixing.
3.5 Create a 1st build.
3.6 Review 1st build with stakeholders.
- Compare 1st build with change request.
- Create a verification report.

3.7 Fix what was wrong on the first build.
3.8 Update use case model.
3.9 Update database.

Repeat from step 3.4 to 3.8 the times of builds we want to create for bug fixing and change request.

(At least two builds are suggested per week).

3.10 Keep writing supporting documentation.
3.11 Plan for training. Consider:
End users, system maintainers, and support staff.
3.12 Refine deployment site and deployment plan. Consider:
Running new system in parallel with previous system in testing mode (if applicable).
3.13 Convert operational databases
3.14 Train End users, system maintainers, and support staff.

If new features have to be added, the iteration is similar to one in the construction phase, requiring analysis, design, and so on.

4.0 Controlling.
4.1 Status reports (weekly).
4.2 Review of the last build.
Product acceptance test (final release).

5.0 Closing
5.1 Use case model updated.
5.2 Design documents updated.
5.3 Operational database.
5.4 Supporting documentation including training for end users, maintainers, and support staff.
5.5 Formal acceptance documentation of the system signed by the sponsor, user or customer (even all stakeholders).
5.6 Lessons learned document.
5.7 Post-mortem assessment

3. Conclusions

There are different causes that make software development to fail. One of these factors is the poor project management. In this work we presented how to minimize the factor of failure related to project management by the elaboration of a guide. The guide provides assistance to a project manager who is in charge of a small software project.

The approach of this work integrates best practices of the PMBOK Guide and CMMI software engineering standards within a RUP-oriented software development cycle. PMBOK and CMMI offers guidelines to consider within a project and a software project respectively. There is similitude between some CMMI Level 2 (Staged) key process areas and PMBOK processes, e.g. Project Planning key process area and Planning Process Group in the PMBOK Guide. This similitude allows us to match certain requirements of the CMMI Level 2 (Staged) to processes of the PMBOK Guide. This approach allows managers, project leaders, and even a novice to software development following the best practices of project management. The suggested plans on this guide covers points from PMBOK Guide and CMMI Level 2 (Staged) for every phase of the software development.

PMBOK Guide and CMMI are not itself software development cycles. For this reason, the RUP phases were selected as a software development approach. The RUP approach is iterative, architecture-centric, and use-case-driven. The RUP has iterations in every phase of its development. Each iteration builds on the work of the previous iterations to produce an executable that is one step closer to the final product. PMBOK Guide best practices in combination with CMMI were placed into every iteration of the RUP phases. As a result, managerial activities support RUP with changing documents, risk identification and addressing on early phases of the project, defect detection and correction over several iterations, and doing integration during the development and not at the end.

Identification of risks is emphasized on the guide, as well as commitment of the participants. Identification of risks allow us to detect possible causes of deviations from the established plans and adjust our planning according to the situation; that is, budget and time frames are controlled and evaluated in every iteration and phase. In this work, the contact with stakeholders is common due to the continuous reviews and feedback from them, so the expectations of the stakeholders are covered.

References

[1] Jurison, Jaak. 1999. “Software Project Management: The Manager’s View” Comm. of

Assoc. for Information Systems, vol. 2, article 17. August 13th, 2005 from http://cais.isworld.org/articles/2-17/default.asp?View=html&x=33&y=6.

[2] Page-Jones, Meilir. “Praktisches DV-Projektmanagement”. Germany (Carl Hanser) 1991.

[3] Charette, Robert N. “Why Software Fails” IEEE Spectrum, vol. 42, no. 9, pp. 36-43, September 2005.

[4] Software Engineering Institute. (2005). “What is CMMI?”. October 6th, 2005, from http://www.sei.cmu.edu/cmmi/general/general.html

[5] Kroll, Per, Kruchten, Philippe. “The Rational Unified Process Made Easy”, USA, Pearson Education Inc, 2003.

[6] Schwalbe, Kathy. “Information Technology Project Management”, 4th ed., Canada, Thomson Course Technology, 2006.

[7] Kroll, Per. (2004). “Dr. Process: How many iterations should you have in a project?”, developerWorks Rational. January 20th, 2006 from http://www-128.ibm.com/developerworks/rational/library/434.html

[8] Spence, Ian, Bittner Kurt. (2004). “Managing iterative software development with use cases”, developerWorks Rational. January 27th, 2006 from http://www-128.ibm.com/developerworks/rational/library/5093.html

[9] American Science Institute of Technology. (1997-2006). “Transition”. January 30th, 2006, from http://www.amscitech.com/_common/_topics/UML/transition.htm