Appendix A

Drafting the Development Plan

Objectives

Upon completion of this appendix, you should be able to:

- Describe how the SunTone Architecture Methodology relates to constructing a development plan
- List the resources (namely developers) and the skills required to build the system
- Select use cases for each iteration based on a set of criteria
- Document the development plan using a UML package diagram

Relevance

Discussion – Some of the hardest decisions in constructing software are: Where does the team start? How often should a team *release* the system during development? The development plan solves these issues by providing a road map to the construction of the system.

- How do you (or your project manager) decide how many developers are required to build a system?

  _____

  _____

  _____

- How do you (and your development teams) decide what functionality to build first?

  _____

  _____

  _____

- Have you worked on a team that used iterative development? What benefits do you see to this strategy?

  _____

  _____

  _____

Additional Resources

Additional resources – The following references provide additional information on the topics described in this appendix:

- Booch, Grady. *Object Solutions (Managing the Object-Oriented Project).* Reading: Addison Wesley Longman, Inc., 1994.
- Brooks, Frederick. *The Mythical Man-month (anniversary edition).* Reading: Addison Wesley Longman, Inc., 1995.
- Jacobson, Ivar, Grady Booch, James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley. Reading. 1999.

Process Map

This appendix describes a step in the Implementation workflow: creating the development plan. Figure A-1 shows the activity and artifact discussed in this appendix.

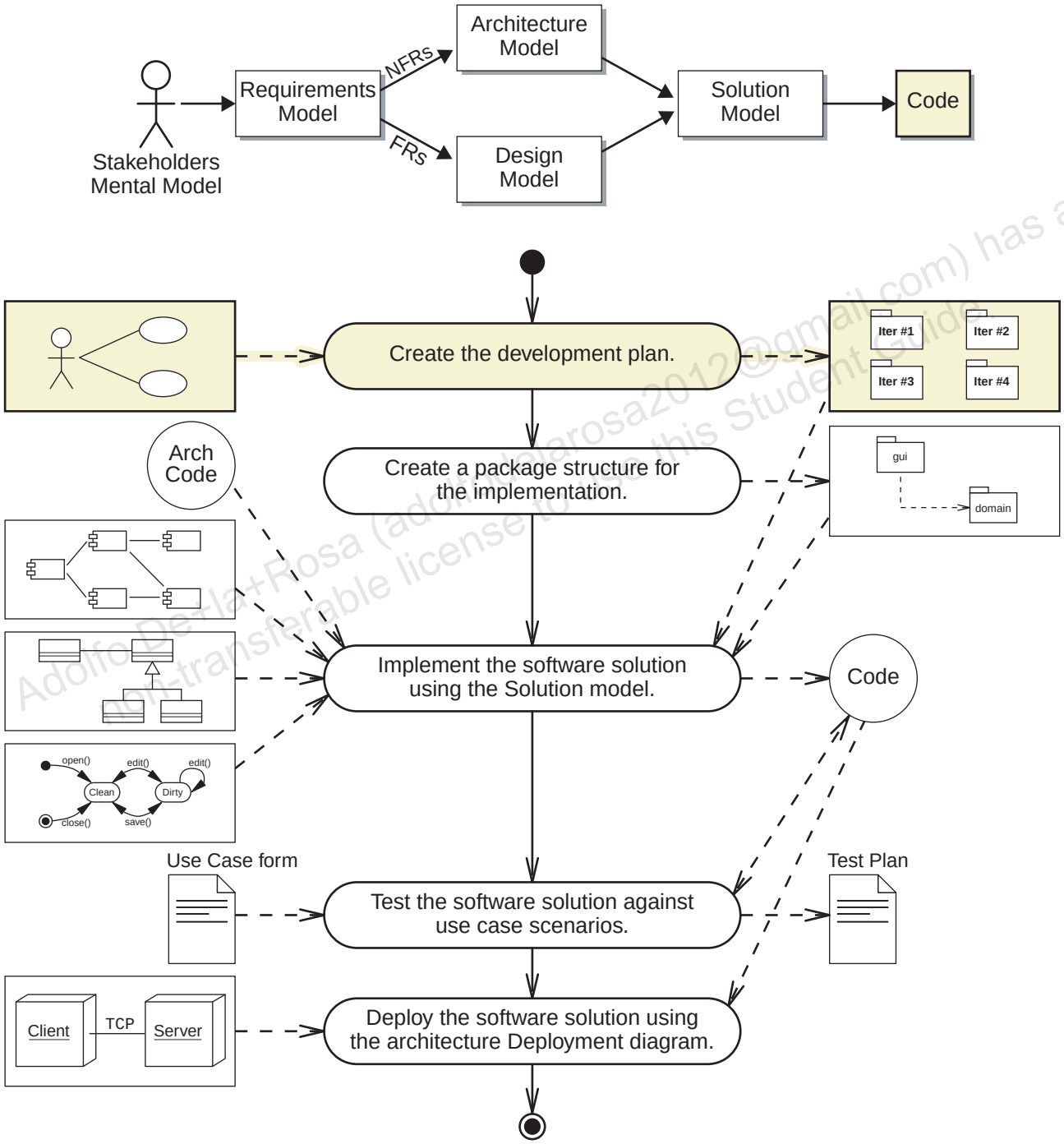

Figure A-1 Implementation Workflow Process Map

Describing How SunTone Architecture Methodology Relates to the Development Plan

SunTone Architecture Methodology is compatible with the Unified Process. Relative to creating a development plan, these are the most important features of the methodology:

- Four phases: Inception, Elaboration, Construction, and Transition

  The life cycle of the project is split into four large phases. During the Inception phase you define the vision of the system. During the Elaboration phase you define the bulk of the system requirements and develop the architecture baseline. During the Construction phase you build the Beta system. Finally, during the Transition phase you deploy the production system.

- Iterative and Incremental

  In SunTone Architecture Methodology, development of the system is done incrementally starting with a small working system. This small system, called the Architecture baseline, is built by the architect (or the architect's team) during the Elaboration phase.

  The Construction phase has multiple, small iterations. Each iteration implements a small set of use cases. Each iteration ends with a working, integrated system. By keeping the iterations short, the development team and client see steady progress.

The development plan will partition the work of the development team into distinct iterations within the phases. Therefore, the plan is partitioned into phases and iterations.

Creating a development plan is challenging because:

- Every developer has their own speed.

  Developers are not interchangeable parts. Every person has different abilities, experience, and development speed. Estimates should be made by the developers themselves. This provides better estimates to the project manager. This also provides buy-in from the development team.

- The size of the team affects individual performance; larger teams add communication overhead.

  Small, focused teams tend to be most effective. Small might mean five plus or minus two developers. When a team becomes too large, the inter-team communication becomes a significant time issue.

However, some projects might have large, separable subsystems that can be developed in parallel by separate teams; each of these teams should be as small as possible.

- Project complexity adds to hidden infrastructure that is not obvious from the use cases.

  Being use-case-driven is only half the story when it comes to building a complex application because use cases only identify the functional requirements. A complex system might have significant non-functional requirements. The NFRs must be considered during planning as well as the FRs (embodied in the use cases).

  The purpose of the Architecture baseline is to identify and mitigate the complexity and risks of this hidden infrastructure.

- Estimates and project schedule should be reevaluated periodically.

  In an incremental project, the project schedule must be viewed as a changeable, *living* document. The project manager should review and update the schedule periodically, and at major milestones, such as at the end of the Inception and Elaboration phases.

Listing Needed Resources and Skills

Based on the FRs and NFRs from the SRS, the architect determines how many people are needed on the team and the skills required to develop the system.

- Development team size is determined by the size and complexity of the project.

  The greatest factors in determining the size of the development team are the size and complexity of the proposed system. However, it is important to realize that it is not so easy to increase the size of the team to reduce the length of the project life cycle. (see Brooks chapter 2)

- Skill set is determined by the architectural needs of the project.

  The architecture of the system will determine the skills that the development team must possess. For example, if the architecture requires database access, then at least one team member must have SQL experience.

These issues are discussed in more depth in the following sections.

Determining the Developers

Factors that determine the number of developers:

- Size and complexity of the use cases

  As mentioned previously, the size and complexity of a use case can be a factor in determining the team size because a complex use case tends to require more developers to build that part of the system in the short time frames devoted to an iteration.

  However, a large use case might be split across multiple iterations if there are separable parts (or flows) to the use case. For example, use case E6 “Generate Reports,” might include a dozen or so distinct reports. The creation of the set of reports could be split across multiple iterations.

- Size and complexity of the architectural baseline

  If the required infrastructure is large or complex, then more developers might be needed to build that part of the system in the short time frames devoted to an iteration.

- Issues with legacy system integration

  A project that requires integration with a legacy system must have one or more developers devoted to this task. These developers should have experience with the legacy system.

- Duration of the project

  If the project is calendar-driven, then the team size might be increased to reduce the overall project schedule. However, you cannot increase the project team without complicating other issues such as inter-team communication, which will tend to extend the duration of the project.

- Modularity of the components

  Several developers can work on different components if the architecture partitions the system into many small modular components. If the architecture is not very modular, then these components must be worked on by a single team.

  Also, separation of components within specific architecture tiers facilitates clear job roles: business component developer, web component developer, web content creator, GUI developer, data access developer, and so on.

For the Hotel Reservation System, the following factors influenced the size of the development team:

- The size and complexity of the use cases is relatively small.

  Only a few use cases were large or complex, such as E1 “Managing Reservations” and E5 “Managing Reservations Online.” These two use cases can also be split up into smaller activities, which makes partitioning the work easier.

- The project is not schedule-bound, so the duration of the project can remain flexible.

  The Hotel Reservation System project does not have a fixed deadline, so the duration of the project can be flexible. This means that a smaller team can be used to build the system.

- The system can be separated effectively into three general technical areas:
  - GUI development
  - Web application development
  - Business logic (plus infrastructure) development

Based on this information, the development team for this project will include one GUI developer, one Web application developer, and one business component developer.

Determining the Skill Set

The skill set of the team is important to the success of the project. Factors that determine the skill set are based on high-level architectural decisions:

- Programming language and platform

  Will the project be developed in an OO language? Has the SRS constrained the specific language for the project? What operating system (OS) or hardware platform will be used? The answers to these questions will determine elements of the team's skill set.

- Specific technologies:

  Likewise, does the SRS (and the architecture baseline) constrain the technologies with which the team must be skilled? Based on the different architectural tiers, here is a sample of technologies that might be needed for a project.

  - Client tier: HTML, JavaScript, Swing, VB, and so on
  - Presentation tier: servlets, JSP, PHP, Perl, ASP, and so on
  - Business tier: RMI, EJB, CORBA, and so on
  - Integration tier: JDBC, XA, MOM, and so on
  - Resource tier: DBMS, EIS, and so on

- Development environment (tools, IDE, and so on)

  What tools will the team use? This is often in control of the development team (and its management), so this is not as critical.

Here is an example set of skills for the Hotel Reservation System project:

- J2SE (v1.3) is the selected platform

  Java technology will be used throughout the project. The Java platform was selected to provide a high degree of portability as well as great n-tier technologies such as web components and RMI.

- Client tier:

  There will be three applications: WebPresenceApp, KioskApp, and HotelApp. The first two are web applications and the second is an intranet GUI application. These are the required technologies:

  - HTML (for WebPresenceApp and KioskApp)
  - Swing (for HotelApp GUI)
- Presentation tier: servlets and JSP pages

  The WebPresenceApp requires Presentation tier (web) components. This will be Java technology servlets and JavaServer Pages.
- Business tier: RMI

  RMI technology is used to communicate from the Client and Presentation tiers to the Business tier. The components on the Business tier will be standard Java objects.
- Integration tier: JDBC/SQL

  The Integration tier will require JDBC technology and SQL to communicate to the database.
- Resource tier: PostgreSQL DBMS

  The Resource tier will use a relational database. The Open Source PostgreSQL system will be used.

Selecting Use Cases for Iterations

Each iteration should deliver a clearly defined portion of the behavior of the complete system. Therefore, complete use cases should be assigned to specific iterations. The following are factors that help determine how to partition use cases into iterations:

- Determine the criteria for grouping use cases into iterations.

  Each project will have different criteria for prioritizing use cases. This is described in the next section.

- Estimate the development duration for each use case.

  The development team must estimate the construction time for each use case. These estimates must be done by the developers themselves, because every developer has her own speed.

- Estimate the recommended length of each iteration.

  Estimating the length of an iteration is largely based on how many iterations the client wants. Some clients might want to see numerous iterations if they are concerned about the project staying on track. Other clients might be totally comfortable with the development team and require fewer iterations.

  Typically, the length of an iteration is between a few weeks to a few months.

- Determine which use cases will be handled in which iteration within which phase of the project.

  Use cases are assigned to iterations based upon the available resources and the prioritization of the use cases.

Partitioning use cases is described in more detail in the following sections.

Criteria for Prioritizing Use Cases

Each project will have different criteria for prioritizing use cases. The following are a few factors to help you determine the priority of each use case:

- Risky use cases should be handled as early in the project as possible.

  The SunTone Architecture Methodology service recommends all risky use cases be developed before the end of the Elaboration phase.

- Architecturally significant use cases should be handled as early as possible.

  An architecturally significant use case is one that presents a technical risk to the project. The most common technical issues include: concurrency requirements, timing or performance requirements, and legacy integration requirements.

- Essential use cases should be handled as early as possible.

  The priority of the use case from the SRS also helps to determine how to group the use cases into iterations. Essential use cases should be handled very early in the development. These should take precedence over High-value and Follow-on use cases, except if a High-value use case is also risky.

- High-value and Follow-on use cases can be distributed in the Construction phase, with Follow-on use cases being handled last.

  Leave the use cases that are deemed unimportant to this version of the system to the end of the project.

There might be other reasons to give a use case a higher priority; projects can be as flexible as possible. This is an important side-effect of using iterative development.

Estimating Development Time for Use Cases

Each development team must be able to estimate the development time for a given use case. The following are a few factors to help you understand how to approach estimations:

- Efficient tier separation (a developer for each major tier function)

  It helps to have clear tier boundaries (client tier, application tier, and integration tier) for a project. Typically, a team will have a single developer for a given tier. This configuration is useful because it enables the team to focus on a specific area of expertise, such as GUI development or database integration.

Note – An alternative strategy is to assign a use case to a single developer. This developer would implement the use case throughout every tier: client/presentation, business, integration, and resource. This strategy works well when every developer on the team has all of the required skills to work in all tiers.

- Skill and *speed* of the individual developers

  Every developer has a certain level of experience; the more experience the developer is the faster the developer can code. However, some developers are inherently faster programmers than others. Speed and skill are the two main factors for an individual developer to estimate how fast that developer can develop a given use case.

- Complexity and amount of infrastructure development

  Not all programming is confined to implementing the functional requirements of a use case. Some programming involves building the infrastructure within which the functional components must fit. The amount of infrastructure development will likely be higher in the beginning of the project than later in the project.

- Number of scenarios in a use case

  The number and complexity of the scenarios for a use case can help identify the difficulty of the implementation of the functional components for the given use case. For example, the scenarios might identify how many different paths there are through the user interface. This might identify how many screens are involved or how many major functions in the business services need to be coded.

For the Hotel Reservation System, use case "E1: Manage Reservation:"

- Is partitioned into four tiers: Client, Business, Integration, and Resource

  The client tier will be a Swing GUI. The application tier requires service components to coordinate the use case functions as well as a variety of entity components for the domain objects. The integration tier requires the development of data access components to persist the entities within the database. The resource tier requires the specification of the database tables for the entities.

- Requires significant infrastructure development

  Because this will be the first use case to be developed, there will be a significant amount of infrastructure development. On the client tier, the fundamental security mechanism must be implemented to support login. On the application tier, the RMI server must be implemented. On the resource tier, the PostgreSQL DBMS must be configured and populated with the initial schema.

- Includes three major scenarios (create, update, and delete)

  On the client tier, it is likely that a single GUI window could be used to implement all three scenarios. On the business tier, the service component must have all of the CRUD functionality to manage reservation entities as well as to create and retrieve customers.

For the Hotel Reservation System, use case "E1: Manage Reservation:"

- GUI developer: 6 weeks

  The GUI developer will spend the first week prototyping the fundamental GUI layout that will be used for all HotelApp screens. The second week will be spent developing the main screen and the login dialog box. The remaining weeks will be spent developing the reservation form and any other related dialog boxes for this use case.

  The breakdown of tasks include:

  - GUI layout design: 1 week
  - MainUI screen: 1 week
  - ResvUI, CustMgtUI, and PayMgtUI: 4 weeks

- Web developer: N/A

  This use case does not contain a web interface, so there is no work for this developer.

- Business (and integration) developer: 4 weeks

  This developer, Annie Codewalker, is a programming wizard, and she has estimated four weeks for the following: RMI service component for the use case, entity classes, DAO classes, database schema, and RMI server.

  The breakdown of tasks include:

  - Three remote services: 2 weeks
  - Five entities: 0.5 weeks
  - Five (partial) DAOs: 2.5 weeks

These durations are determined by the two developers by estimating the number of components required by the given use case. This information can be determined from the Solution model.

Determining the Duration of Each Iteration

Each project will have different criteria for determining the duration of an iteration. The following are a few factors to help you determine the duration:

- Long enough to develop the longest use case

  Any given iteration must be long enough to include development time for the longest use case in that iteration.

- Short enough to see results and maintain the project's momentum

  Iterations should be kept as short as possible to show results to the client and to build momentum within the team.

- Iteration Duration = For each UC, sum of UC duration

  As a rough estimate, the duration of an iteration is the duration of each use case specified for that iteration.

- UC Duration = For each developer, maximum development time for the use case

  The duration for a use case is the maximum development time of all developers working on that use case.

Table A-1 provides an example estimate for the first iteration of the Hotel Reservation System.

Table A-1 Developer Estimates for Iteration 1 Use Cases

Use Case	**GUI Developer**	**Web Developer**	**Business Developer**
E1: Manage Reservation	6 weeks	N/A	4 weeks
E5: Manage Reservation Online	N/A	4 weeks	1 week
Developer Workload	6 weeks	4 weeks	5 weeks

Each developer works in parallel, but a given developer cannot work on two use cases in parallel. So for example, the GUI and web developers can work in parallel, one on E1 and the other on E5. The Business component developer needs to spend four weeks on E1 and then one week on E5 for a total time of five weeks. Therefore, iteration 1 will take six weeks total.

Grouping Use Cases Into Iterations

These are a few factors to help you determine how to group use cases into iterations and phases:

- Group risky and architecturally significant use cases into iterations in the Elaboration phase.

  During the Elaboration phase, you construct the Architecture baseline and eliminate the risks in the projects. The architecturally significant and risky use cases must be scheduled in the iterations of the Elaboration phase.

- Group all (remaining) Essential and High-value use cases into iterations in the Construction phase.

  As mentioned earlier, schedule all Essential and High-value use cases as early in the Construction phases as possible.

- Leave the Follow-on use cases for the end of the Construction phase.

Figure A-2 shows an example schedule of iterations in the Elaboration and Construction phases for the Hotel Reservation System.

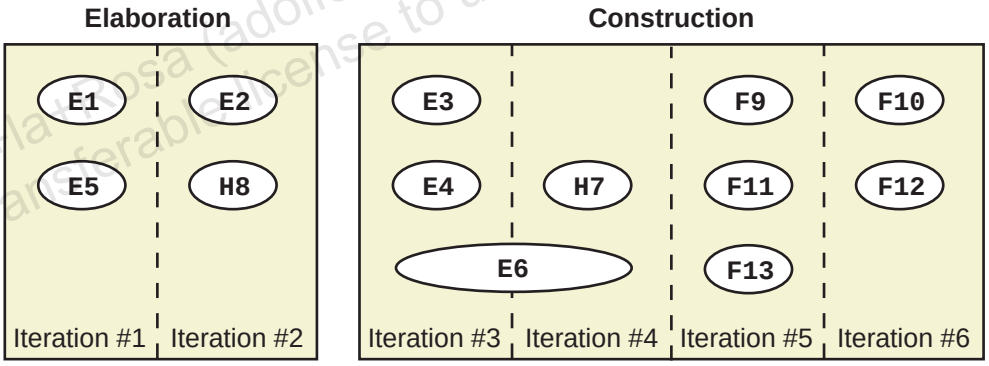

Figure A-2 Iterations for the Hotel Reservation System

Documenting the Development Plan

There are many ways to document the development plan. Most project managers use a tool that support GANTT charts. However, these are often detail-oriented. A UML package diagram can be an alternative:

- A UML package diagram shows the iteration and phase groupings.

  You should provide the customer with a high-level view of the iteration schedule. This can be done pictorially by providing a UML package diagram of the use cases for each iteration.

- The UML package notation.

  A UML package icon can represent a group of other modeling elements. Figure A-3 shows two variations on the package icon.

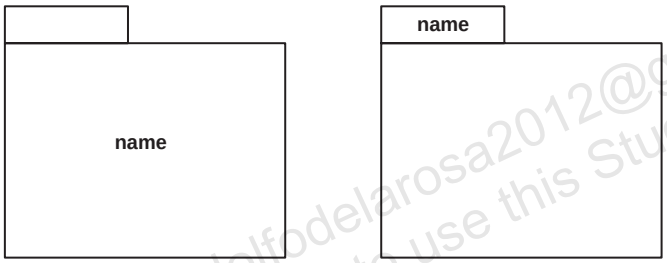

Figure A-3 Two Variations on the UML Package Icon

  - The package name can be placed in the body box or in the name box.

    Use the first view (with the name in the body box) when you do not want to show the contents of the package, but just want to name the package. Use the second view (with the name in the name box) when you do want to show the contents.

  - You can place any UML entity in a package, including other packages.

    The contents of a package icon are completely unrestricted. You can include single UML entities, partial diagrams, complete diagrams, or even nested packages.

Figure A-4 demonstrates the use of a UML package diagram to provide a high-level view of the schedule for the Elaboration phase for the Hotel Reservation System.

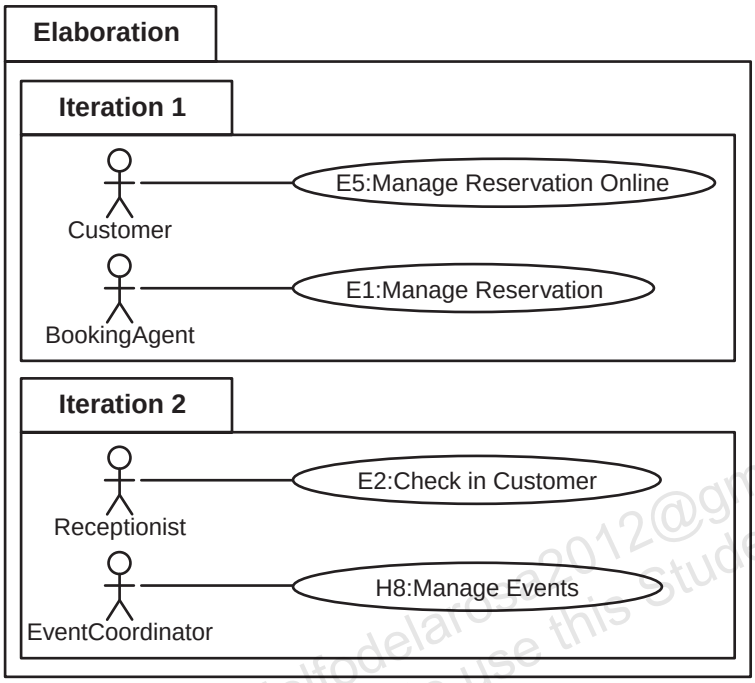

Figure A-4 Package Diagram of the Elaboration Phase for the Hotel Reservation System

Summary

In this appendix, you were introduced to a technique for estimating a use case development effort as well as scheduling iterations. Here are a few important concepts:

- The members of the development team are constrained by project size and the skill set required for the technologies used by the project.
- Development is organized into iterations. Each of which acts as a mini-project.
- Use cases are partitioned into iterations. The duration of the iteration is determined by the time estimates for each developer for each use case.
- The development plan can be represented using a UML package diagram showing iterations as packages containing the use cases to be developed in that iteration.