1

SOFTWARE PROJECT BASICS

INTRODUCTION

Human endeavor, from its earliest hunter/gatherer roots, was carried out in teams, each with a hierarchy of roles. As civilization progressed, the need for structure and rules increased. A large farm is a team organization based on a simple hierarchy of an owner, overseers, and employed laborers. The Industrial Revolution created factories which required more complex hierarchies, both within teams and between teams. Factories aggregated the production of goods for consumption into concentrated units capable of greater productivity. To achieve this great jump in productivity, rules were developed to effectively run the factories. These developments were the genesis of the art and science of managing production, which has been called *production management*.

Classification of organizations. The type of production can be used to classify organizations based on the *manner* in which goods are produced. The categories are:

- Mass production: continuously produces the same products
- Batch production: produces goods in batches; each batch is similar, but not identical
- Flow process production: production of chemicals, pharmaceuticals, and fertilizer products, generation of electricity, etc.
- Job order production: produces tailor-made goods (i.e., goods are produced only when an order is received)

Initially, management texts focused on *mass production*, *batch production*, and *flow process production systems* (also known as "made to warehouse" production systems). In *made to warehouse* production systems, goods are produced and stored in warehouses for distribution. The significant feature of mass production and flow process production is that the rate of consumption/demand *equals or exceeds* the rate of production for the product. In batch production, the rate of production *exceeds* the rate of consumption/demand for the product. The goal of production management is to balance both rates.

Production management texts, however, did not address organizations such as ship building, aircraft manufacturing, heavy equipment manufacturing, etc. These organizations are known as job order production or *made to order* organizations. In *made to order* organizations, items are produced only after an order is received.

By leaving out job order "shops," management texts also excluded organizations that constructed buildings, highways, and other infrastructure facilities. These types of organizations are certainly not serial production organizations even though they create wealth and employ people. Their work was classified as *projects*. Some knowledge, however, was gathered and released under the title of *project management*. Job order production system organizations latched onto this concept and became project-based production systems.

Presently, management theory addresses organizations in two basic categories: production organizations and service organizations. The art and science of managing these organizations has metamorphosed from *production management* to *operations management*.

Similarly, we can categorize organizations by the *nature* of their operations:

- Continuous operations: organizations with fixed facilities that carry out *similar* operations day after day continuously and produce products for stockpiling in warehouses (real or virtual)
- Project operations: organizations with fixed but flexible facilities that carry out *dissimilar* operations from day to day and produce only against a customer order

More and more organizations are moving toward project operations due to market forces, which put emphasis on individual preferences while reducing costs. Gone are the days of the famous words of Henry Ford, Sr.: "You can have the car of any color as long as it is black."

The project operations category has seen significant development over the past few years under the title "mass customization." Mass customization blends aspects of continuous and project operations.

Having put the concept of project operations in an historical perspective, see Table 1.1 for a comparison of continuous operations with project operations. Mass customization walks the line between the two extremes identified in Table

Table 1.1. Comparisons of Continuous Operations with Project Operations

Item Number	Aspect	Continuous Operations	Project Operations
1	Product design	Designed once: updated as needed/dictated by market forces	Designed for every order received
2	Trigger for commencement	Marketing asks for the product	Customer's order triggers commencement
3	Planning	Periodic: annual, quarterly, monthly, weekly, etc.	Order-wise as well as periodic
4	Workstation design	Low cost: to produce one type of component	Potentially higher cost: versatile workstations to produce a wide variety of components
5	Required education levels for staff	Low: needs to understand instructions and can be easily trained (leads to a flatter training curve)	High: needs to be able to interpret drawings/ instructions and may require longer training (leads to a steeper learning curve)
6	Products	Batches of identical products	Products range from similar to (but never identical to) to radically different
7	Types of workstation operations	Mostly repetitive with little variety	Mostly nonrepetitive with wide variety
8	Specialization	Highly feasible	Limited specialization
9	Planning	Planning utilization of facilities becomes more predominant and important	Planning development of the product becomes predominant, while facility utilization planning becomes less important
10	Organizational structure	Hierarchical mostly	Mix of hierarchical and matrix organization
11	Customers	Repetitive customers possible to a high degree	Normally one-off customers with low probability of repeat order for same product

1.1, typically with most of the benefits of each, but with a greater reliance on self-directed teams that make hierarchies and matrix organizations very nervous.

Description of a project. Let's now examine what comprises a project: a project is a temporary endeavor with the objective of manufacturing (producing or developing) a product or delivering a service, while adhering to the specifications

of the customer (including functionality, quality, reliability, price, and schedule) and conforming to international/national/customer/internal standards for performance and reliability. Translation:

- A project is a temporary endeavor.
- A project has a definite beginning and a definite ending.
- No two projects will be identical, although they may be similar.
- Each project needs to be separately approved, planned, designed, engineered, constructed, tested, delivered, installed, and commissioned.
- A project may be stand-alone or a component in a larger program.
- A project is executed in phases, with an initiation phase and one or more intermediate phases and a closing phase.
- Many projects have a transition phase (e.g., handover to customer).
- A project may extend through a maintenance phase.

A software development project (often shortened to *software project*) has the objective of developing a software product or maintaining an existing software product. Software development projects have several general attributes, including:

- The project has a definite beginning and a definite end.
- The project deliverable is functional software and related artifacts.
- Activities that may be included in a project are user and software requirements, software design, software construction, software testing, acceptance testing, and software delivery, deployment, and handover.
- Activities not included in a project are the activities of project selection/acquisition and post-handover.

Some of the more unique attributes of software development projects include:

- The primary output is not physical — in the sense that the primary deliverable is functional software and no tangible components are delivered — almost everything is inside a computer.
- Process inspection does not facilitate progress assessment — functional software or at least the code is the real measure of progress. In a manufacturing organization, one can see semifinished goods. The proof of work being performed is in the noise made by machines. In a software development organization, visual assessment is not enough to ensure that a person is performing. One needs to walk through the code being developed to ensure that the person is working.
- Despite significant progress in software engineering tools and diagramming techniques, they do not rise to the level of precision of the engineering drawings used in other engineering disciplines.

- Professional associations in software development and standards organizations have not defined standards or practices for developing software as has occurred in other engineering practices. The International Organization for Standardization (ISO) and the Institute of Electrical and Electronic Engineers (IEEE) have defined a number of standards, but these standards are not at the same level of granularity as other engineering standards.
- Although significant improvements in software development methodologies have been made, these methodologies are still largely dependent on human beings for productivity and quality. Tools are available to help in development or testing, but they still have not been able to rise to the level set by the standards and tools used in fabrication/inspection/testing in other engineering disciplines. In other engineering disciplines, tools are available that shift the onus for productivity/quality from human beings to the combination of tools and process. Most would agree that an average-skilled person can achieve higher productivity/quality with tools than a super-skilled person without tools.

Therefore, the rigor of planning is all the more important in software development than in other engineering projects — planning is a critical tool to keep a project focused. In other engineering projects, a simple schedule based on PERT/CPM (Program Evaluation and Review Technique/Critical Path Method) would suffice, whereas in software development projects, increased rigor and more planning documents are required (planning documents commonly required are described in subsequent chapters).

TYPES OF SOFTWARE PROJECTS

Software development projects (SDPs) are not homogenous. They come in various sizes and types. Some examples will help us gain an understanding of the breadth of SDPs:

- An organization desires to shift a business process from manual information processing to computer-based information processing. This project will include studying the user requirements and carrying out all of the activities necessary to implement the computer-based system
- An organization desires to shift a business process from manual information processing to computer-based information processing. The organization does not want the software be developed from "scratch." It wants to use a commercial off-the-shelf software (COTS) product. This

project will include implementation and perhaps some customization of the COTS product to make it appropriate for the organization.
- An organization has a computer-based system that needs to be shifted to another computer system because the existing system has become obsolete and support to keep the obsolete system in working condition is no longer available. This project could include porting the code, training users, and testing the new implementation.
- An organization has a computer-based system and desires to shift it from a flat file system to a RDBMS-based system (relational database management system). Activities will include data conversion in addition to other activities.
- An organization has a computer-based information processing system and needs to effect modifications in the software or add additional functionality. Activities include adding functionality and making required modifications in the software of a third party (if required).
- An organization has developed a computer-based information processing system and wants to get it thoroughly tested by an independent organization. Activities will include testing and interfacing between the organizations.

These examples barely scratch the surface of the breadth of software projects — and new project types keep coming in. In all cases, however, the projects concern software, but the tasks, activities, and therefore the work in each of the projects are vastly different.

CLASSIFICATIONS OF SOFTWARE PROJECTS

Software projects may be classified in multiple ways (Figure 1.1). For example, software projects may be classified as:

- Software development life cycle (SDLC) projects
  - Full life cycle projects
  - Partial life cycle projects
- Approach-driven software development projects
  - "Fresh" development (creating the entire software from "scratch")
  - COTS product customization/implementation
  - Porting
  - Migration
  - Conversion of existing software to meet changed conditions such as Y2K and Euro conversion

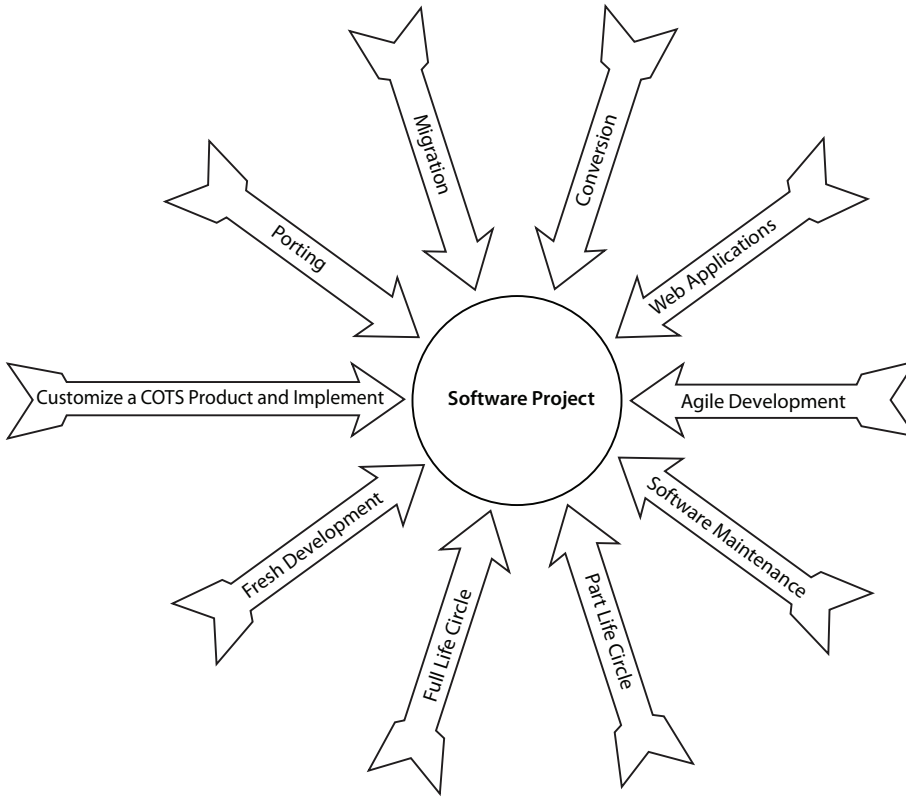

Figure 1.1. Software project types.

- Maintenance projects
  - Defect repair
  - Functional expansion
  - Operational support
  - Fixing odd behavior
  - Software modification
- Web application projects
- Agile development projects

Let's now discuss each type of software project in greater detail.

Based on Software Development Life Cycle

Full life cycle projects. A full life cycle project is a project that traverses the entire arc of the methodology being used: starts at the beginning and ends at

the end. One problem when discussing a full life cycle project is that there is no standardization concerning what constitutes a software development life cycle (SDLC). Generally agreed is that user requirements analysis, software requirements analysis, software design, construction, and testing (regardless of what they are called) are parts of a SDLC. Some of the components of an SDLC that remain in question include:

- A feasibility study determining whether the project is worthwhile
- Special testing that is beyond unit testing, integration testing, system testing, and acceptance testing
- Implementation, including installation of hardware, system software, application software, etc.
- Software commissioning, including creating master data files, user training, pilot runs, parallel runs, etc.

In many instances, when the end product is used within the same organization, these four components are considered part of an SDLC. Alternately, in other circumstances, these components are excluded for organizations that specialize in software development and/or develop software for use by a different organization (unless contractually included or part of a software as a service architecture).

In this book, we exclude these four components. We assume that a full life cycle project is one that starts with user requirements and ends with the delivery of software. Therefore, all post-delivery activities and pre-user requirement activities are not considered to be within the scope of this book.

Partial life cycle projects. Partial life cycle projects are those that include only a portion of the SDLC. In partial life cycle projects, any number of permutations could occur, including:

- Testing projects in which the scope of the work involves conducting the specified or necessary software tests on the software product to certify the product (Unit testing and code walk-through are normally *not* included in this type of project.)
- Independent verification and validation (IV&V) projects in which projects go beyond mere testing, including code walk-through and other forms of validation to determine the efficiency of coding
- A project divided between two or more vendors based on the specialty to derive the advantages of best practices developed through specialization which can lead to defining the project by phase or by combination of phases, such as:
  - Requirements analysis
  - Design
  - Software construction
  - Testing

Approach Driven

Fresh or new software development projects. Fresh or new software development projects are identical to full life cycle development projects previously discussed.

COTS product customization/implementation projects. Numerous popular COTS products are available in the marketplace. Examples include the implementation of ERP (enterprise resource planning software, e.g., by SAP and PeopleSoft), CRM (customer relationship management), SCM (supply chain management), EAI (enterprise applications integration), and data warehousing software. Typical phases in these projects include:

1. Current system study: a review of the present system
2. Gap analysis: a comparison of the current system to the COTS product
3. Customization report: a discussion of the desired levels of customization of the system
4. Statement of work: definition of the required customization of the COTS product
5. Design: how the software will accomplish the task
6. Construction and integration
7. Testing
8. Custom code integration: integration of the code bases (in some cases it can include building a layer over the COTS product and integration of custom developed code into the source code of the COTS product)
9. COTS source code modification (rare)
10. Implementation
11. Training: instruction of users (all classes required) in usage of the system, troubleshooting, and operations and maintenance of the system
12. Transition of the system

Many variations of these phases are also possible for COTS projects.

Porting. Porting projects deal with moving software from one hardware platform to another hardware platform. Porting projects can include:

- Changes in programming language
- Differences between implementations
- Manual intervention to make the existing software work on new hardware without issues

Project execution work in a porting project involves:

1. Documenting the differences between the two versions of the programming languages
2. Developing a software tool to make corrections in the code based on the details mentioned above (Sometimes, vendors of the programming language supply this type of tool.)
3. Execution of the software porting tool to make all possible corrections
4. Manual correction to make any specific corrections needed
5. Conducting the specified software tests
6. Modifications to the software engineering documents required to reflect the changes made in the software
7. Conducting acceptance testing
8. Delivery of the software

Migration. Oftentimes, new versions of programming languages and databases are released. For example, Visual Basic has gone through many versions: from version 1 to 6 and then the release of the next set as 2003, 2005, and 2008. Similarly, Oracle has gone through upgrades: up to version 11. Operating systems have also been upgraded. For example, Microsoft has had many upgrades including MS-DOS, Windows, 2 and 3, and then 95, 98, 2000, XP, Vista, and now Windows 7. When upgrades are released, upgrading software may become necessary:

- To take advantage of new features and facilities provided in the newly released version
- Because an older version is no longer available when additional hardware or system software is installed or the existing software does not function well on the new software (In these days of multitier Web-based software architectures, an upgrade of any tier may necessitate migration!)
- Because limitations existing in an older version are removed in the new release and the existing software needs to be upgraded to remove the limitations

Upgrades are typically due to the ever-changing environment and the increasing needs of an organization. Of course, if the configuration of hardware and software remains exactly same, and the existing software is meeting the user's needs, the software would not need to be upgraded. A new version, however, could contain additional features and facilities that are totally absent in an older version. Therefore, a software tool cannot be used to make the changes that are necessary to port the software. To take advantage of new facilities and features available in the newer version, manual changes are typically required and involve:

1. Studying the new version
2. Deciding which new features are desirable and need to be implemented
3. Developing a functional expansion design document detailing the new features being implemented in the existing software
4. Running and upgrading the software (if an upgrade tool is provided by the vendor)
5. Implementing the functional expansion design in the software coding and incorporating necessary software changes (may also include correcting the existing code)
6. Conducting all the tests necessary to ensure that the software delivers all the functionality it was supposed to before migration and all the functionality that is designed for the new software
7. Conducting acceptance testing and delivering the software
8. Data migration involving (sometimes the project scope may include data migration):
   - Mapping the old database schema to the new database schema
   - Developing software/locating tools provided with the new database (if any) to migrate data from the old database to the new database
   - Running the tools to migrate data from the old database to the new database
   - Arranging for data entry in the new database for those fields that are absent in the old database, but present in the new database
   - Testing the database for known cases using the software, comparing the results with the desired results, and making necessary changes so that the new database is correct
   - Integrating the database with the software

Specific migration projects may have different activities from the activities described above.

Note: Porting and migration projects are similar. There is no strict distinction between the two. Therefore, these two terms are sometimes used interchangeably.

Conversion. Year 2000 (Y2K) and Euro conversion projects are excellent examples of conversion projects. Using a Y2K project as example, the work includes verifying all programs for code limitations and then making any necessary modifications. Typical activities in a conversion project include:

1. Studying the existing software and specifications of the necessary conversion

2. Preparing a conversion guidelines document detailing the procedure for incorporating the required modifications in the software
3. Developing a tool (if feasible) to automatically incorporate the modifications in the software
4. Running the tools or hand coding the changes
5. Performing a manual walk-through of each program to locate the remaining required modifications and implementing them
6. Conducting unit testing (and other tests as specified or as necessary)
7. Conducting acceptance testing
8. Delivering the software

In Euro conversion projects, some countries that did not make use of decimals in their financial software had to incorporate decimals as well as provide for the use of the Euro symbol.

Maintenance

Software maintenance projects are major money makers for software development organizations that are dependent on outsourcing. We need, however, to relax the specific beginning and ending requirements to call software maintenance a *project*. In a software maintenance project, generally there is a contract between the parties to take care of a specific application for a given period of time, e.g., 1 or 2 years, but a contract can be extended as long as both parties remain satisfied with each other's performance or as long as the application is in commission. An overall contract would specify:

- Billing rates
- Mode of requesting work
- Service level agreement (SLA) specifying the priorities and turnaround times
- Persons authorized to initiate/authorize work requests, accept deliveries, give clarifications
- Escalation mechanisms
- Billing cycles and payment schedules

This list could go on forever, depending on the specific needs and the "pain" of the organizations.

Normally, a maintenance work request (MWR) triggers software maintenance work. An MWR can be known by other terms, depending on the organization:

- Program modification request (PMR)
- Program change request (PCR)
- Defect report
- Software change request

Again, this list can also go on forever.

Contractually, an MWR is expected to have proper authorizations and to have them in advance. However, for an immediate need, a telephone call, a fax, or an email can also be used and later regularized through raising an MWR, i.e., post-facto (although frowned upon as potentially leading to loss of control).

Work included in a software maintenance project is classified into five types: defect fixing, operational support, fixing odd behavior, software modification, and functional enhancement.

Defect repair. Defect fixing work involves fixing a reported defect. A defect may be classified as:

- Critical (a "show-stopper")
- Major (hinders smooth functioning of work)
- Minor (mostly a nuisance; work is not affected)

Typically, defect fixing has an associated SLA in which the turnaround time for each class of defect, based on priority, is defined (i.e., the time between when a defect is reported until the time it is fixed, the regression test is completed, and the software is handed over to production). Sometimes, the turnaround time can be as little as hours or minutes, depending on the application and the needs of the organization. Normally, the maximum turnaround time for fixing a defect would be about 2 days. In a defect-fixing scenario, follow-up and progress reporting are frequent and close together. Generally, the steps in fixing a defect include:

1. Studying the defect report
2. Replicating the defect scenario in a development environment
3. Studying the code
4. Locating the defect
5. Fixing the defect, conforming to code change guidelines
6. Arranging for peer review and implementing feedback (if any)
7. Arranging for independent regression testing and implementing feedback (if any)
8. Delivering the fixed code to production for implementation in the production system
9. Closing the request

Functional expansion. When additional functionality is required in existing software, functional enhancement is the tool to achieve it. Functional enhancement work is generally of longer duration and may range from a calendar week upward. Work included in functional enhancement includes:

- Adding a new screen or report
- Adding additional processing functionality (e.g., quarterly/half yearly/yearly processing)

- Adding a new module in the software
- Integrating with another software
- Building interfaces with other software
- Adding new hardware and building an interface to the new hardware in the existing software

Functional expansion generally fits the full SDLC model in which the project leverages the full software engineering process and the project management process and can be treated as an independent project if the duration is sufficiently long enough. The level of process rigor required is typically driven by risk. Each organization has a different definition of a project that should be treated as a functional enhancement project. For example, in one organization, a functional enhancement project is defined as "work with the duration of one person-month of effort or more," while in another, the definition of a functional enhancement project is "40 hours of effort."

Operational support. Operational support is similar to defect fixing. Many times, operational support requires immediate attention. Activities under operational support include:

- Running periodic jobs (end of day/week/month)
- Taking backups
- Restoring from backups
- User management functionality (including creation, deletion, and suspending of user accounts and changing access privileges, etc.)
- Providing "hand-holding" assistance at a specific workstation
- Extracting data and producing an ad hoc report on an urgent basis
- Providing a temporary patch so that operations may continue
- Investigating operational complaints

Again, the list of activities is long and varied.

Fixing odd behavior. In large, complex software systems, and in systems that have been in existence for many years and have undergone software maintenance (e.g., defect fixes, software modifications, and functional expansions), random defects may often crop up under some circumstances, but not in others. These random defects are generally difficult to replicate in a development environment. One reason is because the defect occurs in the field and the person witnessing the defect does not note the chain of events that caused the defect. So until the defect becomes chronic, it might have been handled as an operational support activity and not have been recognized as defect. Such puzzling defects can be placed in the odd behavior category of software maintenance. Odd behavior can be caused by the application software or the system software, a client workstation or a virus,

network security, or a combination of all of these. Diagnosing and correcting odd behavior issues may take longer than a week because correcting odd behavior is similar to conducting research. General steps in fixing odd behavior include:

1. Studying the odd behavior report
2. Trying to replicate the behavior scenario in a development environment
3. Studying the code
4. Listing all possible alternative reasons for the reported behavior
5. Reviewing the code for each alternative for possible opportunities for improvement
6. Iterating/eliminating all causes, one by one
7. Fixing all possible opportunities for code improvement
8. Arranging for peer review
9. Arranging for independent regression testing
10. Delivering the software to production for implementation of improved code in the production system
11. Waiting for another report of the identical odd behavior and repeating all the above steps.
12. Keeping the request open through a period of observation

Software modification. Software modification work is the bulk of software maintenance in most organizations. Modification of working software is necessitated due to:

- Changes in requirements mainly due to changed conditions occurring over a period of time
- Changes in business processing logic
- Convenience for users
- Changes in statutory requirements

Often, modifications include changes to reports, changes to screens by moving around data fields, adding or deleting a data field or two, or some other small enhancement. Steps in the process of software modification include:

1. Studying the software modification request
2. Analyzing the existing software to identify components that require modification
3. Preparing a design modification document and obtaining approval from appropriate executives
4. Implementing the approved design modification in the code
5. Arranging for peer review of the modified code

6. Arranging for independent functional testing of the modified functionality — to ensure that it conforms to the approved design document — and implementing feedback (if any)
7. Arranging for independent regression testing and implementing feedback (if any)
8. Delivering the modified artifact to production for implementation in the production system
9. Closing the request

Web Application

Web projects refer to Web-based application development projects. Web projects differ from other projects because they have more than two tiers:

- Presentation tier
- Database server tier
- Application server tier
- Web server tier
- Security server

A Web application consists of:

- HTML pages that include graphics to enhance the "look and feel" of the Web pages
- Backend programs for data manipulation
- Middleware programs for application server or rules engines
- Middleware programs for security management
- Other application-specific programs

Another notable feature of Web applications is that backend programming and middleware programming may be in different programming languages and may require persons with different skill sets, even for the same project. Another request is for independence from databases and Web browsers, which necessitates coding routines that are not oriented toward functionality. Additionally, a Web application needs to be developed so that it facilitates an easy change of code. Environmental changes that have nothing to do with the organization, e.g., a new security threat, the release of a new browser, or the upgrade of an existing browser, etc., can also trigger software maintenance in a Web application — even though the functionality remains unaltered. Web-based and client server projects have a very similar profile.

Agile Development

Agile software development refers to a group of software development methodologies based on iterative development, in which requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. (Agile project management is discussed at length in Chapter 11.)

CONCLUSION

Software projects are basically projects with a definite beginning and a definite ending, except that the final end product delivered is not physical. Software projects come in various types and sizes. Product maintenance in software is also treated as a project — unlike physical product maintenance. This chapter defines software projects as well as enumerates the different types of projects, laying a foundation for better assimilation of the science and art of software project management. Subsequent chapters will deal with the subject of software project management, building on this foundation.