CHAPTER 1

Introduction

What Is Software Estimation?

Estimation: It is the mark of an instructed mind to rest satisfied with the degree of precision which the nature of a subject requires, and not to seek exactness where an approximation may suffice.
—Aristotle, 330 BC

Estimation is a frequently occurring phenomenon in our everyday lives. When we leave home for the office in the morning, we roughly estimate the time it will take to reach the office. When we plan to make a business analysis presentation in a meeting, we estimate the time it will take to complete the presentation, perhaps including time for questions and answers as well. A building contractor estimates the schedule and cost required to construct a building according to specific requirements. The driver of a car moving at a specific speed encounters an object crossing the road and instantly estimates the time it will take the car to reach the point of intersection and then corrects the speed accordingly. In all these everyday situations, the estimating activity happens sometimes consciously and sometimes subconsciously. It remains for the user to determine the level of estimation accuracy needed, based on the criticality of the activity itself. Above all, the process of estimation itself is refined iteratively in each of the preceding situations, based on historic data or past experience.

The parameters that define estimation vary depending upon the activity being estimated. Activities could be cost, resources (for example, materials), manpower, and equipment. Other parameters could be elapsed time, schedule, or other similar attributes. The key parameter common

to all situations is experience. Whatever the method of estimation, the differentiating factor between a good and bad estimate is the experience of the estimator in arriving at the right mix of the parameters and their attributes. Organizations that are established in implementing quality practices like ISO 9002 and/or SEI/CMMI [1] document individual experiences as best practices and make them available to other estimators across the organization. Basically *the environment in which the activity is being executed* defines the estimation parameters. Typically, we would not be able to correlate and compare the estimations for two different categories of activities. For example, the time required to travel a certain distance in a certain traffic environment cannot be compared with the time taken to complete the construction of a building.

The Dream Project

Sunil was completely shattered. He was totally exhausted, both physically and mentally. He felt his career as a software professional was doomed. After toiling for almost nine months on one of the most grueling software project assignments he had ever had, the project itself was scrapped by the customer. He was totally at a loss to imagine what had gone wrong.

Although the project started nine months ago, to Sunil it seemed just a few weeks ago. Sunil was a lucky project manager indeed; he had been selected to lead a very prestigious software project. Almost all the components of the project seemed to be right. The project was being executed for one of the company's largest customers. The customer team was tech savvy. The technology was state-of-the-art. The application being developed was expected to bring a dramatic increase in business for the customer, based on some very innovative thinking by the business users. Above all, Sunil felt confident this would be a successful project due to the excellent environment in which he was working. His manager was a very good senior software professional, and the team members were well skilled in the technology on which the application was being developed. All the required tools and other development environment were in place.

Sunil's team followed all the processes needed in good software project execution. The requirements were done with due diligence

and after a couple of thorough reviews, the customer signed off on the project. The design was carefully done, keeping both function and performance in view. The team included good developers and the code generated was of good quality. Early tests of some of the modules showed good results. Everything seemed to be going fine, Sunil recalled.

Of course there were a few hiccups on the way, but Sunil thought he addressed them quite well. Although there were a few slippages in the schedule, Sunil was confident of recovering. But now, as he recalled the events that happened during the early days, he wondered whether he really did a good job of fixing the issues and the connected risks. Did he fail to recognize the early warnings? He remembered each of these situations quite vividly:

Project Kick-Off: Immediately after the customer signed off on the big contract for the project, Sunil decided to do a detailed evaluation of the scope first including derived effort and schedule estimations. He brought a couple of senior team members and did an effort estimation based on the past experience of delivering similar projects. They had not previously sized a project of this magnitude; however, they had completed smaller projects and thought they would be able to do an extrapolation and arrive at a good guesstimate for the current project. They quickly realized that it was indeed a very large project. After several rounds of deliberations, it was decided to fix the delivery schedule at 15 months with an average team size of 30 members. The team was aware that the customer had a tight deadline and any further extension of schedule might not be allowed.

Schedule Renegotiation: During discussions with the customer on project execution strategy, team members realized that the customer was quite inflexible on the 10-month delivery schedule agreed upon during contract negotiations. Sunil had no option but to return to the drawing board and recalculate the project schedule and the related impact on other project delivery aspects. They had to do several review rounds to arrive at the best possible project execution plan. The reduction of schedule by 30 percent (from 15 months to 10 months) had a direct impact on the average team size, increasing it by a whopping 50 percent (from 30 members to 45 members). In order to reach this timeframe, they also reduced

the buffer that had been built in earlier. As a result, all the phases of the software project were very tightly planned with little scope left to maneuver slippages.

Requirements Phase: During the requirements gathering phase, the scope was reviewed with the business users. The business users pointed out that due to recent changes in government regulations, the software application required specific changes in order to comply with the new regulations. This also meant changing the functional workflow of activities. Sunil's team included all the relevant changes and did a quick re-estimation. The overall effort had gone up by another 6 percent. Once again, a negotiation with the customer to increase the schedule was denied. The result was an increase in team size by another three members. Sunil was worried about the possible impact and increased risk but was confident his team would deliver.

Design Phase: The first serious signs of problems surfaced during design phase. The software was being developed on the Java platform. The technology vendor had recently introduced a new product that would dramatically change the way the business workflow could be manipulated directly by the end users. Although the product was a boon to the users, it meant two significant deviations to the project team: the team had to learn and understand features of the new product, and the team would need to invest additional effort to include the new product into the revised architecture of the software application. Overall, the effort went up again. Sunil's team was rightly worried this time. The size of the project was ballooning, but the end date of delivery was not allowed to change.

Project Progress Review: Sunil felt it was time to escalate the situation to his manager internally. His manager decided to do a complete review of the project status. Another surprise was soon to be discovered. The project had already slipped by two weeks. The late changes in project scope and the last-minute changes due to the addition of the new product had made their impact on schedule slippage. This situation was further aggravated when it was discovered that the new product was still in beta and had quite a number of bugs that needed to be removed by the vendor. By now the team size had grown to 50 members and it was looking almost impossible to meet the 10-month schedule.

Build Phase: During the coding stage, one of the key senior members fell ill and had to take two weeks leave. Two other developers decided to quit, thus adding salt to the wound. The early schedule slippage had a crunching impact on the time provisioned for the coding (build) phase. Despite Herculean efforts, the team found it impossible to meet the original deadline. Sunil had no other option but to go back to the customer to negotiate more time.

During discussions with the customer, the critical nature of the whole project emerged. The business users had planned strategically for this project to be ready in 10 months in order to meet a possible window of business opportunity that would last only a few months. Missing the deadline would mean a loss of business opportunity. The chances of similar opportunity occurring in the near future were almost nonexistent. Sunil was a depressed project manager when he returned from the meeting with the customer. The project team met to discuss the situation. Many enthusiastic younger team members felt there still was a last chance to meet the deadline if everyone pitched in with extra effort. The team decided to work extended work hours and weekends. There was an air of expectation and things seemed to improve.

But not for long. During early tests, serious performance issues were detected. Inconsistencies in coding standards followed by different groups within the project team were also found. This again was a setback that had a direct impact on delivery schedule. It was now impossible to meet the deadline under any circumstances. After internal deliberations with the team and Sunil's manager, it was decided to convey the sad news to the customer. The only option was to extend the delivery schedule by three months. The customer was aghast at the proposal. The matter was escalated to higher management and after a couple of rounds of senior level meetings it was decided by the customer to scrap the project.

What really did go wrong?

For a detailed solution to this project problem, please refer to Chapter 9, "A Sense of Where You Are."

Ingredients of a Good Estimation

Unless all the key ingredients of estimation are identified and assessed thoroughly, the process of estimation itself will be incomplete and in many situations, the end result will not be of much use. The key elements are discussed here.

Activity Scope

The important key element that forms the basis of estimation is the scope of the activity being estimated. *Scope* is a very loose term and will take different shapes in different situations. For example, if you were constructing a building, the scope would probably be the square feet of area being built. If you were traveling between two locations, the scope would be the distance in miles. If you intend to build a wardrobe for your bedroom, the scope would again be the area in square feet. If you are building a software application, the scope could be the size of the software in terms of functionality that it delivers or in terms of lines of code delivered. If you are making a business presentation to your customer, the scope could be the content that you intend to cover in order to make a good sales pitch. For a sportsperson who is practicing hard to win a 100-meter race, the scope would be the exact 100 meters the person needs to cover.

Work Environment

The environment in which the activity is being executed makes a huge impact on the overall estimation. While driving a compact car in the U.S., I could cover 10 miles in 12 minutes on Highway 101 in California. But it takes 45 minutes to drive seven miles on a similar highway between Madiwala and Electronic City in Bangalore, India. Likewise, it would be as tough to drive on the main streets of New York as it is easy to drive around the Parliament House in New Delhi. Obviously there is a big difference in environment between these situations. In this sense, *environment* refers to the layout of roads, traffic rules, and the people who follow (or do not follow) traffic regulations. It also refers to the capacity of the roads, which can handle a certain density of traffic. In another example, I could be constructing a building of a certain size in a crowded locality of a city or I could be constructing the same building in a jungle area. In the first case, there would be severe restrictions on movement of material and equipment, and in the second case, quite a

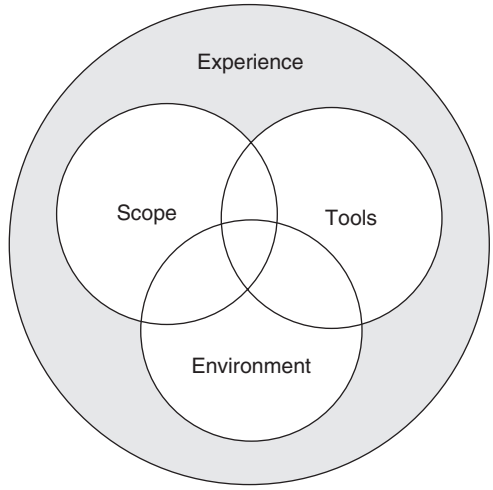

Figure 1.1 *Estimation ingredients.*

bit of effort would go first into clearing the jungle and then making long trips (for the material and for me) to the site on regular basis. Environment topics could range from weather conditions to political interference. They all play a crucial role in the final estimation. Figure 1.1 shows the various ingredients of estimation.

Consistency

Historic data, the competency of the team that executes the activity, and best practices developed over the years derive the estimation of time and hence the schedule for the activity to be delivered. Despite the repetition of similar sets of activities involved in completing a particular assignment, each new attempt is different in some way from previous one. Teams encounter new obstructions, issues, and hurdles, which may or may not have occurred in the previous occasion. It is these iterations that make a person or a team "experienced." And by carefully recording and analyzing the experiences, you can add immense value to the final estimation. In software parlance, the rate of delivery of a unit of work is better known as *productivity*.

Usage of Tools

Tools that are used to execute an activity play a significant role in defining the effort and the time taken to complete the activity. Tools can take

different forms, shapes, and sizes depending upon the type of activity. The tool could be the type of car used for traveling between two locations, the construction equipment used for constructing a building, the sports gear used by the athletes, or the utility of other software tools used by the software developer. Effective use of tools has a major impact on productivity.

Learning from Past Experience

The popular definition of an expert, "One who has learned from past mistakes," has much relevance when you have a need for improving a defined process through multiple and continuous iterations. Fortunately, you need not learn everything from your past mistakes; you also can learn from others' mistakes, provided you have a well-defined process to record experiences and mistakes, and devise corrective actions. Each of the iterations of the improvement cycle will consist of the following:

- Define/refine the execution model
- Estimate required parameters based on the execution model
- Implement the defined model
- Continuously monitor during execution
- Measure key parameters of output generated (metrics)
- Analyze metrics, find the area of improvement, and refine the execution model

In each of these iteration cycles, the original estimation at the beginning of the cycle will be validated at the end of the cycle and based on the data collected, the estimation will be refined.

Software Project Estimation

Is estimation for a software development activity different from other estimations? I would say it is similar to any other estimation activity if you know the key parameters required to do an estimation—for example, scope, environment, experience, and tools. Figure 1.2 enumerates the two sides of the same coin. The three ingredients (scope, tools, and environment) are shown in two different contexts: general engineering and software engineering. In both situations, the contribution of all three ingredients is equally significant and plays an important role in arriving at final project execution estimates.

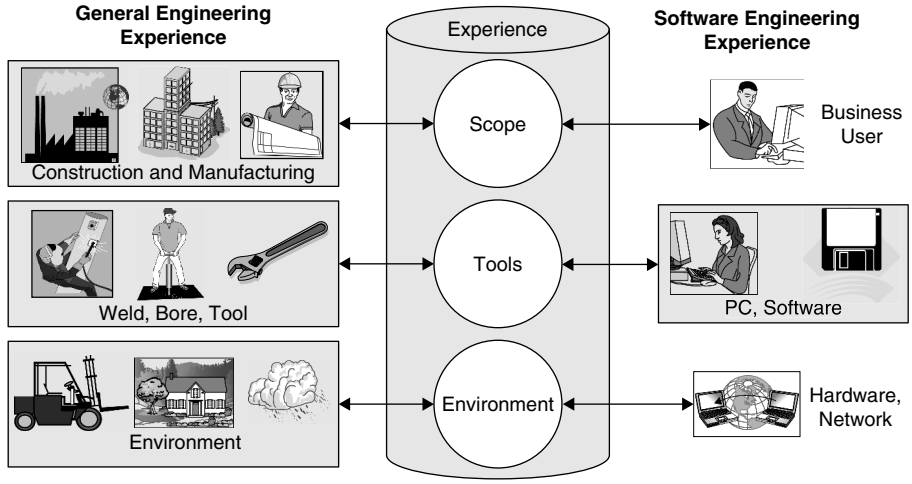

Figure 1.2 *Estimation ingredients—comparison.*

Project Scope

Software developers do realize that it is extremely difficult to capture accurately the scope of a given software project. A wide variety of ingredients make a software system complete. Here are just a few:

- Business functionality addressed through the application system
- The various modules of the application
- The platform, language, and database used
- Tools used as part of the application
- Performance and other execution capacity attributes of the system
- Interface with other systems in the environment

Although the number of methods used to capture the scope of the components of a software system have evolved over the last few decades, almost none of them can size the scope with precision. In software terms, the scope is equated with the *size* of the software system. The size has been defined in different units of measurement, such as Lines of Code (LOC), Function Points (FP), number of programs, and number of objects. Some of the popular software estimation models that have been developed by experts include

- Function Point Analysis Method [2]
- Mark II Function Points [3]

- COCOMO II Model [4]
- Feature Points [5]
- Object Points [6]
- COSMIC-FFP Method [7]
- Delphi Method [8]
- Use Case Points [9]
- Others (See Chapter 14, "Other Estimation Methods.")

Software Environment

A software system needs the right environment to exist and perform in much the same way a fish needs water to live. Most software systems are designed and developed for a particular target environment. Sometimes a variant of the software does execute in similar environments, which have basically originated from a common basic environment. For example, a software system developed for UNIX version 9.4 is likely to work with UNIX version 10.0, and with very few changes, it may work on the Linux operating system as well. But it is very unlikely for a software system developed on a mainframe operating system to work without changes on a Microsoft Windows operating system. There are many such parameters that define the environment in which a software system was developed and as a result, the software system can work only in certain target environments. Some of the key parameters that define the software environment are

- Operating system (including the version)
- Technology platform
- Programming language
- File system
- Database system (if applicable)
- Interfaces to other environments (if applicable)
- Hardware
- Communication system (if applicable)
- Architecture of the application system
- Performance and other scalability expectations from the software system

Developing a software system based on the environment in which the system is expected to execute has a considerable impact on the

estimated effort. In situations where the same software system needs to execute in multiple types of environment, the effort to develop such a feature further adds to the overall effort.

Team Experience

Competency of the developers of the project team is a key factor that impacts the effort required to deliver the software system. Experience gained by developers while delivering software development projects helps enhance the competency of individuals as well as the team. Given the same software project scope, different teams may use different levels of effort. Some of the key aspects that define the competency of a developer team are

- Ability to understand clearly the scope
- Technical expertise on the development platform
- Project management expertise
- Quality procedures and processes
- Software testing skills

Software Development Tools

A wide variety of tools available in the market helps the developer in a variety of ways and forms. Some of the popular tools are

- Design tools
- Build tools
- Tools to review code according to standards
- Online documentation
- Tools to develop repeatable test scenarios
- Configuration tools

Each tool helps in some portion of the software development lifecycle. The benefits of using tools vary. The tools could help you adopt formal designing or coding practices, prepare documentation, configure the development environment, or facilitate easy testing of individual modules or programs. Directly or indirectly, the final benefit of using a tool is that it helps you complete the activity faster, thus saving development time.

The capability of the project team to deliver measurable outputs at a unit rate is termed **productivity.** For example, if the measurable output

of the project is in function points count and the unit of time is hours, the productivity can be described in function points per hour. Experience, environment, and tools are all components of productivity.

Continuous Improvement Cycle

No two software projects are ever the same.
Even if the same project is redone!

Every new software project is a unique experience to the team. I'm sure that software project execution teams will agree with me whether they are part of the internal "Information Systems" department of an organization or they are from an external "Outsourcing" organization, and no matter how many software projects they have executed.

Parameters other than scope, environment, experience, and tools change the way project execution takes place. For example:

- *User* (customer): The adage, "The customer is king," fits very aptly into a software project contract. From concept to commissioning, as the project execution lifecycle progresses, so does the user's conceptual image of the end product. Steve McConnell [11] has observed: "… software development is a process of gradual refinement. You begin with a fuzzy picture of what you want to build and then spend the rest of the project trying to bring that picture into clearer focus." At times the user has only a conceptual image of the product at the start and as time progresses, the image gets refined until the user is almost sure of what he or she wants. And by this time, the software project is almost nearing its end. With the user in this state of mind, imagine the uncertainty the development team would have to manage.
- *Technology:* Upgrades in technology are not uncommon during the project execution lifecycle, particularly if the project is of long duration. And it may be possible that the user has the inclination to adopt the latest technology. If this happens, it has a direct impact on the project execution process and, as a result, the estimation, effort, and schedule.

A good software project execution team understands these fluctuating aspects of project execution and builds a certain amount of flexibility into their execution model.

The estimation effectiveness shown in Figure 1.3 improves with each iteration of project execution. As a process, the learning during each

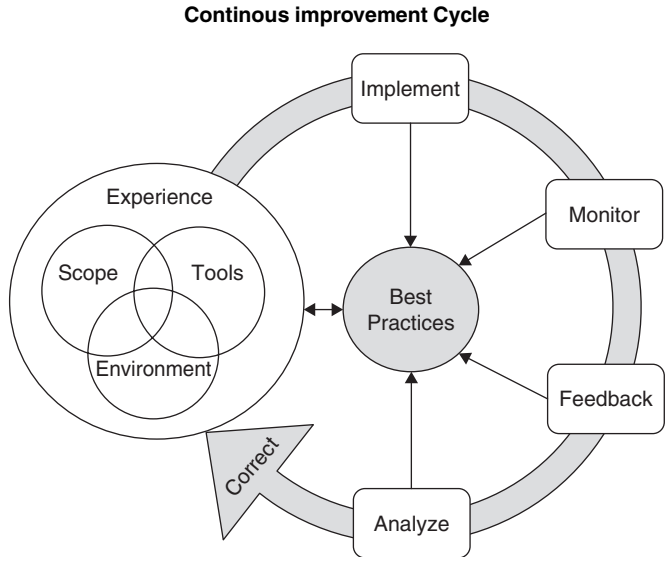

Figure 1.3 *Estimation effectiveness.*

cycle is extracted during the cycle and converted into best practices. The measure and analyze processes help in identifying the shortcomings as well as the best practices that happened during a particular cycle. Based on these findings, a corrective action is applied before the next cycle of project execution.

Why Software Is "Soft"

Have you ever wondered who originally gave the name "software" to applications that were developed using a set of programs in a particular language, and why? Maybe the name originated because the end product was not actually visible to the user and only the effect of the "software" could be experienced. Maybe the name came from the fact that software is malleable enough to be modified, stretched, tinkered with, and purified through fire (destructive testing). The end product could look better and better as it passed through these iterations. Let's look at some of the serious aspects of the *soft* issue here.

Consider a typical software project execution lifecycle. In almost every phase of project execution, the *soft* part of the software being developed causes significant impact on the clear understanding of scope, quality, effort, and finally the schedule of delivery itself. Here are a few parameters to consider:

User: Put yourself in the shoes of the user. You can almost never say with absolute confidence that the scope defined by you, as captured by the project team, is final and that barring a few minor changes there will not be any significant changes in scope as the project progresses. As the user, you know this but confidently ask the project team to proceed with requirements definition. This often happens in project situations, due at least in part to a lax attitude by the stakeholders. The software being *soft* does play a significant role here. The killer assumption you (as the user) make here is that as the software project continues with construction, there exists ample opportunity to modify the scope at any time. The modification could happen even during the acceptance phase!

Designer: As much the user banks on the software being *soft*, the designer also carries the same assumption forward. The design is completed to the extent possible, based on the scope provided by the user, and passed on to the developer team. The understanding is that when additional (or balance) scope is provided, the designer returns to the design document and makes appropriate changes. "But this will mean rework," cries the developer. No problem; the software is *soft* and can be modified.

Developer: The developer team bears the brunt of the entire chain of reactions due to incomplete scope and incomplete design all because of the fact that software is *soft*. This is almost the last phase of project execution before test and acceptance happens. The developer team cannot pass the buck any further. Every small change in scope has a rippling and ballooning effect on the design, build, and test phases. Because of this, the quality, effort, and schedule of the project take a hit. All because software is *soft*.

How does this impact the estimation of the project? It is obvious that impact is huge!

On the other hand, if I compare *soft* with *hard*, there are clear differences in the way *hard* things are produced. For example, imagine

a television manufacturing factory. If you look at an assembly line in the shop that rolls out a particular model of TV, you might be amazed to see the way the shape of the final product slowly builds as it moves along the assembly line. You would also notice that each and every component of the TV belongs in a specific place and it fits perfectly! Why? Someone in the manufacturing design office has taken extreme care to ensure that each and every assembly and subassembly has been carefully crafted to fit. Also planned well in advance are the workflow arrangements—what needs to come in what order and how the parts are tested. Imagine what would happen if you apply the same rules to *soft* products. Can the manufacturer afford to design a circuit board that is in its early stage of draft design and let it go into production? Can the LCD panel of the TV tube or the outer frame of the TV case go into production without the rest of the components being designed to fit perfectly with each other? The daily losses would be huge and the lack of fit among the components would almost turn the set into junk.

But it is *not* so with *soft* products!

Why Software Estimation?

A true professional always plans his or her project meticulously before the actual execution of the work begins. Effort estimation of individual activities is a key input to any good planning process. A building contractor must have a good handle on various activities in different phases of the project with near accurate estimates of resources, material, equipment, costs, etc., in order to be able to monitor and execute the project within a given time schedule. Of course, the customer for whom the building is being constructed must have an estimate of the amount of financing needed for milestone payments. Also as the project execution moves toward the final stages, the re-estimation needs become more frequent and critical. Disruptions in the building construction activities due to reasons like unplanned absenteeism of labor or delay in supply of critical construction material may force the project manager to reassess the impact on project schedules. In order to reschedule the project, it is necessary to re-estimate some of the activities.

Metrics—Past, Present, and Future

Those who cannot remember the past are condemned to repeat it!
—Santayana

Several well-documented analysis reports [10] highlight key factors responsible for failures of software projects, with bad estimation being a major contributor. Going a step further, there are several known reasons for estimations to be bad. These reasons include

- Ignoring past/historic data
- Optimism and bias in estimations
- Uncertainty of requirements
- Non-estimation
- Management pressure
- Unskilled estimators
- Budget constraints

Each of these factors, either individually or in combination, causes bad estimations. Enabling estimators through formal training and mentoring does help, but the training needs to be complemented with a well-defined process to formally collect metrics on past data. If implemented, the development team can now successfully *bat the googlies bowled* (a cricket term) sometimes by the management in the form of budget constraints and unreasonable delivery schedules.

The ISO and the SEI organizations have well-defined processes to collate metrics, and analyze and suggest improvements to a project execution activity. Estimation plays a significant role in defining and capturing metrics as per the ISO and SEI/CMMI recommendations. There are various estimation units that include Lines of Code (LOC), Function Points (FP), and Actual Effort (Person Hours), which can be effectively utilized as a yardstick to measure the size of a software application (these units of measurement are discussed in detail in later chapters). These yardsticks provide a critical reference baseline to rationalize the actual results of the software project execution process across multiple projects. For example, the productivity of the development team is defined as Function Points per Person Month (FP/PM) and sometimes as Person Hours per Function Point. The Defects Density that the development team generates is measured in terms of Defects per 1000 FP (there are other units of measure, too). Organizations that are able to implement effectively the ISO or SEI/CMMI processes find these measuring yardsticks of immense value. They help measure

various project execution parameters across a wide variety of technology platforms, business functions, and even competency of the development teams in an "apples to apples" situation. Here are some of the key benefits of using a metrics collection process in an organization:

- *Past* project execution metrics help in assessing and analyzing the capabilities, strengths, and weakness of the processes, domain, and technology skills as well as the project execution methods deployed across the organization.
- Past experience also shows that as the project execution starts nearing the end stage, the testing and bug fixes activities increase. Typically the programmers in the project team would be under tremendous pressure to manage several activities simultaneously. These activities include completion of ongoing modules, testing the completed modules, and fixing the bugs reported on the tested modules. With the fast-approaching project delivery date, the pressure mounts exponentially, quite often leading to more mistakes. Metrics on these operational issues should be used to fine-tune estimations for similar projects.
- *Present* project execution metrics help in measuring, mentoring, and monitoring the ongoing projects in different lifecycle stages of execution. Comparing them with past metrics helps quick correction and fine-tuning, thus improving the probability of delivering the project on time, within cost, and with quality parameters.
- *Future* needs are those set as targets for achievement in the next six to 12 months. These targets could include improvement of productivity by 5 percent or a reduction in defect density by 10 percent. Once again, good estimation methods with defined units of measurement help in clear definition of future plans.

Estimation Dilemma!

Larry was clearly in a frustrated mood. Despite having the assistance of a good technology expert and couple of senior developers from his team, Larry was unable to pin down the exact development effort that the team needed in order to be able to deliver the project on time.

Just a week ago his manager, Peter, asked Larry to take this new project. The organization was losing considerable revenue because of fraudulent medical insurance claims. In the recent management meeting, it was decided to develop an application that could collate, analyze, and detect possible fraudulent cases well in advance. Peter was given the responsibility to put together a team and deliver this application. Larry had a record of successful project delivery and because this project was critical, his selection was an obvious choice. Larry was given just two days to come up with a project execution plan. Of course, the elapsed time to deliver was decided by the management well in advance. Larry had four months to do this.

Larry had enthusiastically started working on the project execution plan. He arranged a meeting time with a couple of subject matter experts, and obtained the allocation of a trusted technology expert and a couple of senior developers. The team went through an elaborate requirements-capturing process and also a high-level architecture definition of the application. At the end of the second day, Larry had in front of him a fairly well-defined scope. In order to obtain multiple approaches to effort estimations, Larry asked the technology expert and the developers to come up with their estimations separately. As a seasoned project manager, Larry worked on his own estimations. When Larry and his team compared the three estimates, the differences were too high to be in the comfort zone. The lowest and the highest estimates differed by more than 100 percent. Absolute lack of any process to collate and analyze historic data added to the woes of Larry's team. They had no backup support to depend on.

A classical case of lack of estimation capabilities in the organization was staring Larry in the face!

Importance of Estimation

As discussed in the beginning of this chapter, one of the critical factors that determine the success of your project involves how well you are able to estimate the parameters that control the project execution itself. Accurate estimates are as critical to software projects as they are to projects in manufacturing, construction, and similar professions.

Bad estimations or no estimations can lead to situations where the success of the project itself is at risk. Here are a few examples:

- Software projects have a notorious tendency of leaning toward failure if not handled with utmost diligence. Even published reports show success rates of software projects as quite low. Bad estimations are among the major causes for project failures.
- The process of estimation should encompass all the activities that consume effort and provide sufficient contingency for other risk factors that might derail the project. The risk factors include
  - Inconsistent or incomplete project scope definition
  - Competency of the project team resources
  - Complex business rules and algorithms
  - Unexpected change in technology environment
- Project management depends heavily on your anticipating the slippage in the project execution process well in advance and making appropriate corrections. Effort estimates for individual activities and a constant check on deviations in these efforts are critical inputs to good project management practices.

Estimation—Who and How

Software estimation is an art. As such, it requires an estimator to have artistic capability. Each estimator paints his or her own style of estimation picture—some artists follow traditional art forms and others develop their own techniques. No matter what an estimator's style, some of the key stakeholders who need estimates include

- The business folks
- The end user
- The sponsor
- The subject matter experts (SME)
- The project manager
- The developer
- The IT management
- The outsourcing vendor (if involved)

The Business Folks: Without the business people, a software project does not exist, more-or-less. You may have all the other stakeholders you need to execute the project but without a business need it is next to impossible to kick start the project. The business envisions a new business opportunity that exists in the near future. In order to be able to reach and take advantage of the new business opportunity, the business team debates and visualizes the need for their IT group to enhance and provide new business functionality in their application systems. Thus the need for a fresh software project is born. But if you are wondering about the link between business and software estimation, the answer is the time-to-market that is given to the developer team. The clear mandate that is given to the project sponsor is this: No matter what effort and resources are required, you have to deliver by a given time! Any delay beyond that certain stretch of time may result in the whole project becoming inconsequential. The business need may not exist after that point.

The End User: Equally important are the end users. They are the most difficult community of all the stakeholders, and sometimes they are also the business folks themselves. Most end users are typically senior employees of the organization. Having seen several generations of system changes, they are the most difficult to satisfy with further changes to application usage patterns. As such, they need to be involved in key project planning activities from the start. The project delivery schedule is the key area of interest to end users because that determines when they need to start getting acclimatized to the new functionalities in the upgraded applications.

The Sponsor: Being the sponsor of the project brings clear responsibility on two fronts: budget versus expenses, where the sponsor needs to keep a tight check on the costs incurred against the deliveries, and the time schedule the sponsor has promised to the end users (business folks). Without a good handle on estimates of cost and effort, and, as a result, the schedule, it would be impossible for the sponsor to manage these important commitments. The cost estimates also are useful in making a build-or-buy decision while evaluating COTS (Common Off The Shelf) products.

The Subject Matter Experts (SME): Similar to the end user, the SMEs need to plan their participation in advance. The project schedule that is based on estimated effort would highlight the periods of participation by SMEs.

The Project Manager: Project management, among other essential ingredients, is one of the critical factors needed in order for a software project to be successful. Meticulous planning, continuous monitoring, and applying corrective factors at the appropriate time are the traits of a successful project manager. Estimation of size, and hence the effort, is

a major input to project management process. Tom DeMarco [12] has observed, "Estimating is at the very heart of the difficulty we have in controlling software projects." It is not enough if a project manager does the estimation once in the beginning of the project. Re-estimation activity at the completion of every milestone of the project is absolutely essential. We will discuss this in detail in subsequent chapters. Change in scope during project execution is a common phenomenon, better known as *scope creep*. A project manager will need a good estimation model to size the scope creep and arrive at the increase in overall effort, change in schedule, and costs based on the actual impact on the project deliverables due to scope creep.

The Developer: As important as it is to other stakeholders to be aware of estimation needs of various activities of the project, it is equally essential that the developer team have a good understanding of the estimated effort that has been calculated for each of the design or coding activities assigned to them. A true developer will understand the effort allotted to the assigned activity and will continuously check to ensure the target is met.

The IT Management: The responsibility for managing all IT-related activities within the organization remains with the IT group. For them, the main focus is to develop and deliver the project on time and within budget. Also of interest to the IT group are the maintenance costs when the application goes live (Total Cost of Ownership). The estimated effort and hence the schedule and cost would be key inputs.

The Outsourcing Vendor: Quite frequently the software projects are outsourced to external vendors. The vendor prepares estimations based on the skills and competency he or she has in domain and technology areas.

If we look holistically at the complete scenario of a software project execution, the single most important aim that emerges is the successful delivery of the project that meets the scope, is within the budgeted costs, delivered on time, and of good quality. The business user is ultimately the last and the main person to be impacted if there is any slip in the schedule or in the quality of the product delivered.

Conclusion

Estimation is an art of approximation and an activity that is done before the product has even started to take shape. It is natural that measurement of such an activity is never 100 percent perfect. Given that the estimation

of an activity is dependent on four major factors—scope, environment, experience, and tools—it is evident that if you improve the confidence in predicting the outcome of these factors, it helps in improving the accuracy of estimation.

Understanding the process of estimation in software projects is not enough by itself; it is equally critical that you understand how to apply the estimation knowledge appropriately to different situations during the lifecycle phases of the project execution. And while applying the estimation methods, you also need to take care to implement the necessary corrective steps in order to bring the project back on the track. This is experience! You could follow all instructions as written in this book, but bringing the right mixture of the process and methods comes only by experience.

Boehm and Fairley [13] have highlighted the significance of being aware of the context in which estimations are being done in their article "Software Estimation Perspectives." They make the following points:

- It is best to understand the background of an estimate before you use it.
- It is best to orient your estimation approach to the use you're going to make of the estimate.

I will attempt to explain all of these complexities as well as the ambiguities of the process of estimation throughout the rest of this book. Let's explore together the various nuances of software estimation in the coming chapters.

References

1. The Capability Maturity Model Integrations (CMMI) was developed by the Software Engineering Institute (SEI), Carnegie Mellon University. www.sei.cmu.edu/cmmi
2. The Function Point Analysis method was developed by Allan Albrecht and is now maintained by International Function Point Users Group. www.ifpug.org
3. Symons, Charles. *Software Sizing and Estimating: Mark II Function Points* (Function Point Analysis), John Wiley & Sons, 1991.
4. Boehm, Barry W. *Software Engineering Economics,* Prentice Hall, 1981. http://sunset.usc.edu/research/COCOMOII/index.html

5. Feature Points developed by Capers Jones of Software Productivity Inc. is a variant of IFPUG Function Point Analysis. www.spr.com/products/feature.shtm
6. Banker, R., R. Kauffman, and R. Kumar. "An Empirical Test of Object-Based Output Measurement Metrics in a Computer Aided Software Engineering (CASE) Environment," *Journal of Management Information System*, 1994.
7. COSMIC-FFP, version 2.0. Copyright © 1999. The Common Software Measurement International Consortium (COSMIC). www.cosmicon.com
8. Delphi Method was originally developed by the Rand Corporation (1948) and improved into Wideband Delphi Method by Barry W. Boehm and colleagues in the 1970s.
9. Schneider, Geri, and Jason Winters. *Applying Use Cases—A Practical Guide*. Addison-Wesley, 1998.
10. The Standish Group International, Inc. *The Chaos Report*. 1995.
11. McConnell, Steve. *Rapid Development*. Microsoft Press, 1996.
12. DeMarco, Tom. *Controlling Software Projects.* Englewood Cliffs, NJ: Prentice Hall, 1982.
13. Boehm, Barry W., and Richard E. Fairley. "Software Estimation Perspectives." *IEEE Software.* November/December, 2000.

Other Interesting Reading Material

McConnell, Steve. "What Is an Estimate," in *Software Estimation—Demystifying the Black Art.* Microsoft Press, 2006. Pp. 3–14.