Chapter 2
Estimation of Costs and Time for the Development of Distributed Software

Manal El Bajta, Ali Idri, Joaquín Nicolas Ros, José Luis Fernandez-Aleman, and Ambrosio Toval

2.1 Introduction

Most of today's software development organizations aspire to save time and reduce costs. Therefore, globally distributed environment has invaded the software development industry. The strategy of distributed software development generates many benefits that support the development of software product in an effective way, but this strategy still faces many challenges which may hinder the success of globally distributed software development projects. In this context, a significant number of projects failed to deliver within time and budget in globally distributed environment [1]. Thus, managing the globally distributed environment is a key characteristic. However, in order to successfully plan software development projects' activities, it is important to sustain a high level of accuracy to cost and time estimation methods.

Developing software products in a cost-effective way is the overwhelming objective of many organizations. In addition, the ultimate goal is the accurate estimation of the required amount of effort for the completion of each project. Many research studies indicate that projects without realistic planning and accurate estimation are often beyond their allocated budget and the proposed completion time [2–4].

The drivers involved in the distributed environment are investigated with respect to four aspects: (1) software product, (2) personnel attributes, (3) computer

M. El Bajta (✉) • A. Idri
Software Project Management Research Team, ENSIAS, Mohammed V University, Rabat, Morocco
e-mail: manal.elbajta@gmail.com

J.N. Ros • J.L. Fernandez-Aleman • A. Toval
Software Engineering Research Group, Regional Campus of International Excellence, "Campus Mare Nostrum", University of Murcia, Murcia, Spain

© Springer International Publishing AG 2017

Z. Mahmood (ed.), *Software Project Management for Distributed Computing*, Computer Communications and Networks, DOI 10.1007/978-3-319-54325-3_2

attributes, and (4) project attributes [5]. We also suggest that distributed software development projects' success is never isolated to one particular driver.

Although there are many methods and techniques available to assist in creating distributed software project effort estimates, they are still far from the required accuracy. Several authors concerned with software development have given varied suggestions for these inaccuracies and ways to overcome some of them [6, 7]. In contrast, this chapter focuses on ways in which existing effort estimation methods can be tailored to account for global software development. It investigates the influence of the different factors that affect the effort estimation method's accuracy in the context of globally distributed software development projects. Furthermore, this chapter presents the effort estimation methods based on the treated factors.

The chapter is structured as follows: Sect. 2.2 presents the globally distributed environments. Section 2.3 reports the software effort estimation process. Section 2.4 outlines software cost/time estimation techniques for global software development (GSD). Section 2.5 discusses the main cost and time drivers. Section 2.6 presents the risk analysis; finally, the conclusions and future work are presented in Sect. 2.7.

2.2 Globally Distributed Environment (GSD)

GSD refers to software development that is done by multiple teams in different geographic locations. The teams are separated physically, and they are located in different countries within one region or around the world. The teams can either be from one organization or from multiple different organizations (outsourcing) [8].

Global software development is usually considered to be much more difficult than collocated software development given the many different challenges related to the software development in a globally distributed setting. These challenges include negative impact of physical distance, cultural differences, and many other complexity factors which are elaborated in the following subsections [9, 10].

Past studies have shown that tasks take about 2.5 times longer in distributed setting than in collocated setting [11, 12]. Other studies reported that about 40% of GSD projects fail to deliver the expected benefits, due to the lack of theoretical basis and difficult complications in GSD project [13, 14]. On the other hand, Teasley et al. [15] reported that in collocated teams, productivity and job satisfaction are much higher than projects that do locate the entire project team in a war room.

The additional activities and difficulties in global software development require additional effort for substantial planning, coordination, and control overhead in the day-to-day governance of global software development. This additional effort should be considered in the time and cost estimation. Hence the time and cost estimation in GSD is more complex than in local development.

2.2.1 Challenges

Although GSD offers several benefits, the distributed work has also many challenges (Table 2.1). If globally distributed software projects are not managed neatly, then they are likely to turn any company into a loss-making business [16]. That means that there are many challenges associated with global software development. Physical separation among project members has diverse effects on many levels. The following factors have been gathered from research literature [17] to have an impact on the amount of effort and cost required for global software development.:

- *Geographic distance*: Software development, particularly in the early stages, requires much communication, coordination, and control [18]. Geographical distance is a measure of the effort required for one actor to visit another and can be seen as reducing the intensity of communication [19], especially when people experience problems with media and have difficulties finding a sufficiently good substitute for face-to-face interaction [20]. Kraut and Streeter [21] found that formal communication is useful for routine coordination, while informal communication is needed to face uncertainty and unanticipated problems, which are typical of software development. They observed that the need for informal communication increases dramatically as the size and complexity of the software increase. In a large software organization, developers can spend on average up to 75 min per day for informal unplanned communication [22]. In general, low geographical distance offers greater opportunity for periods of collocated teamwork.
- *Temporal distance*: Time zone differs among project members when development team is distributed around the world. Temporal distance is a measure of the dislocation in time experienced by two actors wishing to interact [19]. Temporal distance can be caused by time zone difference or time shifting work patterns and can be seen as a factor that reduces opportunities for real-time collaboration, as response time increases when working hours at remote locations do not overlap [23]. Temporal dispersion reduces the possibilities of synchronous interaction, which is a critical communicational attribute for real-time problem solving and design activities. In practice, teams in different time zones have few hours in the work day when multiple sites can participate in a joint synchronous meetings and discussions. Temporal dispersion can also make misunderstandings and errors more likely to occur [24].

  This leads to delay in response to asynchronous communication. For example, an e-mail sent from one site arrives after working hours at the destination; as a consequence, the response cannot be sent until the next day begins, and it will be visible to the sender only when he/she comes to office on the following day.
- *Linguistic distance*: The lack of a common native language creates further barriers to communication [25, 26]. Linguistic distance limits the ability for coherent communication to take place [27]. English has become the popular language of GSD [28]. This affects not only the quality of communication but also the choice of communication media. Language skills can impede

communication in more subtle ways. When participants to a conversation have different levels of proficiency, the group with better language skills occupies a position of strength and can appear to be more powerful and thus suppress important communication through unintended intimidation [28]. Further, lack of proficiency in the chosen language can lead to a preference for asynchronous communication, which can be an impediment if video and teleconferencing are important communication media [29].

- *Cultural distance*: GSD requires close cooperation of individuals with different cultural backgrounds which often creates another barrier for efficient work. Cultures differ on many critical dimensions, such as the need for structure, attitudes toward hierarchy, sense of time, and communication styles. These differences have been recognized as major barriers to communication. Culture also affects interpretation of requirements; domain knowledge used to fill in gaps or place requirements in context varies considerably across national culture [30]. Culture also interferes with collaboration when cultural norms result in conflicting approaches to problem solving.
- *Social challenges*: Another fundamental challenge in global software development is the social issues like fear and trust. Fear and distrust can negatively impact the motivation, the desire to work, the cooperation, and the communication and share of knowledge with remove colleagues. Hence, it has a direct bearing on the success of implementing global software development [31]. It is very difficult for individuals and groups to trust and build relationships with people they feel threaten their jobs. On-site teams in expensive countries are fearful of their job security when off-site teams are added in less expensive locations; this creates mistrust to their off-site colleagues as well as their own management's motives. This can result in clear examples of not wanting to cooperate and share knowledge with remote [26, 31].

Table 2.1 Challenges in global software development

	Challenges
Temporal distance	Reduced opportunities for synchronous communication
	Typically increased coordination costs
	Management of project artefacts may be subject to delays
Geographic distance	Face to face meeting difficulties
	Lack of critical task awareness
	Difficulties to convey vision and strategy
Linguistic distance	Knowledge transfer will not occur smoothly
	Language confusion and misunderstandings
Sociocultural distance	Cultural misunderstandings
	Reduced cooperation arising from misunderstanding
	Different perceptions of authority can undermine morale
	Adaptation of managers to local regulations
	Impact on coordination caused by inconsistent work practices

In some cases, wherein people have successfully worked together for up to year in a collocated situation, once a virtual team strategy was fully implemented, these problems soon came to the fore.

2.2.2 Benefits

This section identifies the main benefits that have been associated with global software development.

2.2.2.1 Cost Savings

One of the most obvious reasons for organizations to embark on a challenging and risky endeavor such as GSD is, not surprisingly, the potential to reduce development costs. By moving parts of the development work to low-wage countries, the same work can be done for a fraction of the cost [32]. The basis for this benefit is that companies are globalizing their software development activities to leverage cheaper employees located in lower-cost economies. This has been made possible by the deployment of cross-continental high-speed communication links enabling the instantaneous transfer of the basic product at hand: software.

The difference in wages across regions can be significant, with a US software engineer's salary being multiple times greater than that of a person with equivalent skills (at least parts) from Asia or South America. However, this seems to be rising, and there has been hyper-growth in local IT employment markets such as in Bangalore. It is our experience that companies are now looking at alternative locations, which offer more acceptable attrition rates with the continued promise of cheaper labor.

2.2.2.2 Reduced Time

Having developers located in different time zones can allow organizations to increase the number of daily working hours in a "follow-the-sun" development model which can decrease cycle time. Time zone effectiveness is the degree to which an organization manages resources in multiple time zones, maximizing productivity by increasing the number of hours during a 24-h day that software is being developed by its teams. When time zone effectiveness is maximized to span 24 h of the day, this is referred to as the "follow-the-sun" development model. This is achieved by handing off work from one team at the end of their day to another team located in another time zone. The approach can aid organizations which are under severe pressure to improve time to market [11].

2.3 Software Effort Estimation Process

In software project management, effort estimation is the process of developing an approximation of the monetary and temporal resources needed to complete project activities [33]. Usually software is developed in projects, and hence software cost and time estimate can be considered as an approximation of the monetary and temporal resources needed to complete software.

2.3.1 Estimation Process

In order to establish an accurate effort estimate for software, a structured approach with significant amount of work is needed. The software effort estimation can be seen as a small size project which needs to be carefully planned, managed, and followed up. Many organizations have different processes for software effort estimation. These processes vary in many aspects, and there does not seem to be one common process which is used in all organizations and in research. The process for software cost and time estimation data gathered from the NASA's *Handbook for Software Cost Estimation* [34] enables us to develop the following table (Table 2.2). It consists on preparing a description of cost analysis requirements, revising its processes and its procedural requirements document and cost/time estimation handbook accordingly.

Most of the software effort estimation models view the estimation process as being a function that is computed from a set of cost drivers. And in most estimation techniques, the primary driver or the most important driver is believed to be the software size. As illustrated in Fig. 2.1, a view of software estimation process, the software requirements are the primary input to the process and also form the basis for the estimation.

2.3.2 Estimation Accuracy

The effort estimation accuracy helps to determine how well or how accurate our estimation is when using a particular model or technique. In addition to the degree of project determination, estimate accuracy is driven by:

- Level of non-familiar technology in the project
- Complexity of the project
- Quality of reference cost estimating data
- Quality of assumptions used in preparing the estimate
- Experience and skill level of the estimator
- Estimating techniques employed
- Time and level of effort budgeted to prepare the estimate
- The accuracy of the composition of the input and output process streams

Table 2.2 Software cost estimation process from NASA

Number	Action	Description
Step 1	Gather and analyze software functional and programmatic	Analyze and refine software requirements, software architecture, and programmatic constraints
Step 2	Define the work elements and procurements	Define software work elements and procurements for specific project
Step 3	Estimation software size	Estimate size of software in logical Source lines of code
Step 4	Estimate software effort	Convert software size to software development effort
Step 5	Schedule the effort	Determine length of time needed to complete the software effort
Step 6	Calculate the cost and time	Estimate the total cost and time of the software project
Step 7	Determine the impact of risks	Identify software project risks, estimate their impact, and revise estimates
Step 8	Validate and reconcile the estimate via models	Develop alternate effort, schedule, and cost estimates to validate original estimates and to improve accuracy
Step 9	Reconcile estimates, budget, and schedule	Review above size, effort, schedule, and cost estimates and compare with project budget and schedule
Step 10	Review and approve the estimates	Review and approve software size effort, schedule, and cost estimates
Step 11	Track, report, and maintain the estimates	Compare estimates with actual data

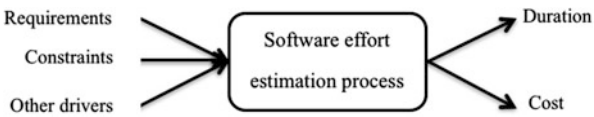

Fig. 2.1 View of software estimation process

We can assess the performance of the software estimation technique by the following two mechanisms:

2.3.2.1 Mean Absolute Error (MAE)

Mean of absolute error (*MAE*) (Eq. 2.1) [35] is computed by averaging the total of absolute errors (*AE*) (Eq. 2.2).

$$MAE = \frac{1}{n}\sum_{i=1}^{n} AE_i \tag{2.1}$$

$$AE_i = |e_i - \widehat{e}_i| \tag{2.2}$$

2.3.2.2 Mean Magnitude of Relative Error (MMRE)

MMRE is defined in Eq. 2.3. This measure is derived from the magnitude of the relative error (*MRE*) as shown in Eq. 2.4. This *MRE* criterion has been criticized by some researchers for being biased toward underestimates, which makes it not significant for being an accuracy measure [36, 37].

$$MMRE = \frac{1}{n}\sum_{i=0}^{n} MRE_i \tag{2.3}$$

$$MRE = \frac{AE_i}{e_i} \tag{2.4}$$

where e_i and $\widehat{e}_i$ are the actual and predicted effort for the ith project.

Each of the error calculation techniques has advantages and disadvantages. For example, absolute error fails to measure the size of the project especially in GSD context, and mean magnitude of relative error will mask any systematic bias (do not know if the estimation is over or under).

2.4 Software Cost/Time Estimation Techniques for GSD

The cost/time estimation has been in the focus of software engineering research for many decades, and hence a high number of different estimation techniques have been developed [38–40]. Unfortunately most of the techniques for software cost estimation have been developed before the recent trend on global software development. Many techniques assume that the software is developed locally, and therefore they do not take into account the additional challenges for the development of distributed software [41, 42].

Estimation for the development of distributed software differs from estimation of local software development at least in two different ways. Firstly, there is a large overhead effort caused by several factors such as language differences; cultural barriers, or time shifts between sites; etc. Secondly, many factors (such as the skills and experience of the workforce) are specific and cannot be considered globally for a project. In many projects, the development sites have very different characteristics, and thus the productivity and cost rate is different between sites.

In the recent research, techniques used to estimate project effort and task duration in distributed context [43] include expert judgment, estimation by analogy, and algorithmic models (i.e., COCOMO II, SLIM, and recently function point analysis-based models) [41].

2.4.1 Expert Judgment

Experts' judgment is one of the methods by which assessors conduct their effort estimation via using their expertise and their logical reasoning to estimate the required amount of effort needed to develop a software product. The accuracy of this method mainly depends on the skills, knowledge, and experience of the assessors to estimate the required amount of effort to complete a given project. Expert judgment can be very accurate, but it fails to provide an objective and quantitative analysis of what are the factors that affect effort and duration in GSD context, and it is hard to separate real experience from the expert's subjective view [44]. The accuracy of the estimates depends on how closely the project correlates with past experience and the ability of the expert to recall all the facets of historic projects.

2.4.2 Estimation by Analogy

Estimating by analogy means comparing the proposed project to previously completed similar project, where the project development information is known. Actual data from the completed projects are extrapolated to estimate the proposed project. This technique is relatively straightforward. Actually in some respects, it is a systematic form of expert judgment since experts often search for analogous situations so as to inform their opinion. The methodology that should be followed to succeed the estimations by analogy involves characterizing the proposed project, selecting the most similar completed projects whose characteristics have been stored in the historical data base, and deriving the estimate for the proposed project from the most similar completed projects by analogy [41, 45].

2.4.3 Algorithmic Models

The algorithmic methods are designed to provide some mathematical equations to perform software estimation. These mathematical equations are based on research and historical data and resort to inputs such as source lines of code, number of functions to perform, and other cost/time drivers such as project effort, design methodology, task allocation, team size, etc. The algorithmic methods have been largely studied and offer several advantages such as generating repeatable estimations, refining and customizing formulas, supporting a family of estimations or a sensitivity analysis, and calibrating previous experience. Models such as COCOMO II (Constructive Cost Model) and SLIM Model are the most frequently algorithmic methods used in a GSD context [43]. In the following, we present:

2.4.3.1 Constructive Cost Model

One of the popular and extensively used algorithmic models for the estimation of cost and schedule of a developing software was given by Shruti Jain [46] and is known as the Constructive Cost Model (COCOMO) [47, 48]. The parameters and equations that are used in this model are obtained through previous software projects. The size of code is usually given in KLOC (thousand lines of code), and the obtained effort is in person months (PM). The PM represents the number of hours that a person spend to complete a given task presented in a calendar month. COCOMO II deals with variety of factors that influence development of distributed software projects' effort estimation. There are three submodels for COCOMO II: Application Composition Model, Post-architecture Model, and Early Design Model. COCOMO II includes factors in order to steer the effort estimation team to make better approximation based on the influencing factors. These factors are related to organizational and team characteristics. Each factor has values from range of very low to extra high rating level. The weight of scaling factors could divert according to organizations and projects. The following are the equations which COCOMO II proposed to estimate the required effort:

$$PM = A \times \text{Size}^{E} \times \prod_{i=0}^{n} EM^{i} \tag{2.5}$$

where:

- n represents the number of drivers in a GSD context.
- $A = 2.94$ (for COCOMO II). size is estimated by kilo source lines of code (KSLOC) measure.
- $E = B + 0.01 \times \sum_{i=1}^{4} \text{Factor}$
- EM represents the effort multiplier; $B = 0.91$ for COCOMO II.

$$\text{Duration} = C \times PM^{D+0.2\times(E-B)} \tag{2.6}$$

where $C = 3.67$, $D = 0.28$, and $B = 0.91$

2.4.3.2 Slim

SLIM [49] is an algorithmic method that is used to estimate effort and schedule for projects. The underlying reason for developing SLIM is to measure the overall size of a project based on its estimated SLOC. It is represented by two equations: Eq. 2.7 for allocating productivity parameter (PP), expressed in man years, which would be required in Eq. 2.8 for calculating effort.

$$PP = \frac{\text{Size}_{SLOC}}{\left(E_{\text{Man, Year}}/B^{1.13}\right) \times \text{Duration}\left(Y^{4/3}\right)} \tag{2.7}$$

$$E_{\text{Man, Year}} = \left[\frac{\text{Size}_{SLOC}}{PP \times \left(\text{Duration}_{\text{Years}}\right)^{3/4}}\right]^3 \tag{2.8}$$

where:

- $E_{\text{Man,Year}}$ represents the amount of effort required to accomplish a given task in a man-year unit.
- Y is the development time in years.
- B is a special skill factor and is based on size and duration.

Muhairat et al. [43] investigated the effects of different factors on the accuracy of effort estimation methods in GSD environments. Precisely, COCOMO II and SLIM methods of estimating project efforts were considered. They discovered that the estimation methods were less accurate in determining the actual time of completion of some software development projects. The main factor that affected this outcome included the project environment. They concluded that developing software in a GSD environment always requires more effort and time to complete.

2.5 Cost and Time Drivers

As already known, the distributed software development introduces new challenges in the software engineering area. In order to have a better project planning for multisite projects, it is important to identify the main drivers that can increase the project's effort. This section aims to describe these main effort drivers and their impacts on a distributed project.

Analyzing the main researches found in the literature and the feedback from project managers about the impact on project duration and effort would enable us to suggest some effort drivers for distributed software development projects. We present the effort drivers extracted from theoretical research and interview analyses. The effort drivers are split into four categories depicted in Table 2.3 [41]: product, platform, personnel, and project factors. Therefore, the effort drivers tend to be measures of system size and complexity, personnel capabilities and experience, hardware constraints, and availability of software development tools.

2.5.1 Product Factors

The product factors are determined by the novelty of the software to be developed. This category factors indicates the degree of innovation which is directly

Table 2.3 Software drivers

Category	Drivers
Product	Code size
	Reuse
	Product complexity
Platform	Design and technology newness
	Time zone
	Platform volatility
Personal	Team size
	Team culture
	Team trust communication
	Development productivity
Project	Project effort
	Project management effort
	Process model
	Task allocation
	Work pressure
	Client involvement
	Work dispersion

proportional to the level of spontaneous communication, the need for specific domain knowledge, and the frequency of unforeseen changes. Another important factor is the work assignments that have to be carefully crafted and taking into account the organizational structure and the functional coupling among software units [50]. Therefore, the architecture has major influence on the efforts needed to coordinate the development phase. Indicators for the degree of architectural adequacy might be modularity, interface match and dependencies, and communicability of the architecture. Examples of product cost drivers of COCOMO II are:

- *Required Software Reliability (RELY)*: This is the measure of the extent to which the software must perform its intended function over a period of time.
- *Date Base Size (DATA)*: Measure to capture the effect of large data requirements have on product development.
- *Required Reusability (RUSE)*: This cost driver accounts for the additional effort to construct components intended for reuse on the current or future projects.
- *Documentation Match to Life Cycle Needs (DOCU)*: Measure of the suitability of the project's documentation to its life cycle needs.

2.5.2 Platform Factors

The platform factor refers to the target-machine complex of hardware and infrastructure software. Platform products have more demanding task characteristics than derivative products. Specifically, platform projects undertake development of

greater levels of new technology and have higher levels of project complexity. Examples of platform cost drivers of COCOMO II are:

- *Execution Time Constraint (TIME)*: This is a measure of the execution time constraint imposed upon a software system.
- *Main Storage Constraint (STOR)*: This is a rating that represents the degree of main storage constraint imposed on a software system or subsystem.
- *Platform Volatility (PVOL)*: This is a measure of the complex of hardware and software.

2.5.3 Personnel Factors

As for the personnel factors, it includes cultural fit mainly related to closeness of team members' mental models [51] which is influenced by the combination of countries involved, the international experience of the teams, etc., skill level measured by educational level and language skills indicating the formal abilities of remote team(s) [50], shared understanding embodied by tacit knowledge that is required indicating the level of completeness of documentation and specification and the common knowledge about goals, and finally information sharing constraints representing competitive restrictions on information distribution, e.g., when working with external subcontractors or in security-sensitive environments. Examples of personnel cost drivers of COCOMO II are:

- *Programmer Capability (PCAP)*: Current trends continue to emphasize the importance of highly capable analysts.
- *Applications Experience (AEXP)*: This rating is dependent on the level of application experience of the project team developing the software system.
- *Language and Tool Experience (LTEX)*: This is a measure of the level of programming language and software tool experience of the project team developing the software system.

2.5.4 Project Factors

Regarding project factors, we might consider the novelty of collaboration model by analyzing the initial cost for the search of offshore partners and contract negotiation. The tools and infrastructure represent the homogeneity of the tool chains used in all sites and potential ramp-up costs for setting up the infrastructure in remote sites and finally the physical distance representing the potential overlaps of working time and, accordingly, the intensity of use of asynchronous communication media and collaboration tools [52]. Examples of project cost drivers of COCOMO II are:

- *Multisite Development (SITE)*: The assessment and averaging of two factors, site collocation and communication support.
- *Required Development Schedule (SCED)*: This rating measures the schedule constraint imposed on the project team developing the software.

2.6 Risk Analysis

In order to analyze the impact of risk involved in the development of software, the project manager has to identify the risk drivers. Software risk components can be classified as "cost" and "time" risks. The degree of uncertainty that the project budget will be maintained is the cost risk. The degree of uncertainty that the project schedule will be maintained and that the product will be delivered in time is the time risk.

Software risks are managerial issues which should be handled through proper management of the project especially when estimating costs and times. Only expert manger associated with software project office can handle these issues, while a less experienced software manager may lead to un-controlling the risks and ultimately result in the failure of the project. Software risks should be monitored and controlled since the starting phases of the project management life cycle [53].

The GSD is becoming very difficult, complex, and challenging in the context of software project management as the user problem is getting more and more challengeable [19, 54]. In this respect, the risk management in distributed software development is also much complex than in local software development. It particularly has specific concerns that may not be obvious until their impact has been realized. Many projects got failed they did not realize, soon enough, the importance of certain common factors in GSD projects [41]. Table 2.4 presents the potential risks in a GSD project and provides their cost and time impact in this respect.

To systematically identify risks and evaluate appropriate risk mitigation for estimating cost and time in the GSD context, we analyze the features of GSD and then elaborate how they are impacted by risks [55].

Efficiency

Software and IT companies need to deliver promptly and reliably while the competition is literally a mouse click away. Hardly any other business has so low entry barriers as IT and therefore stimulates an endless fight for efficiency along the dimensions of improved cost, quality, and time to profit. GSD clearly helps in improving efficiency due to labor cost differences across the world, better quality with many well-trained and process-minded engineers especially in Asia, and shorter time to profit with following the sun and developing and maintaining software in two to three shifts in different time zones. Risks directly related to the efficiency target are project delivery failures, requirement, and design quality (Table 2.4).

Table 2.4 Risk items and their impacts

	Risk item	Cost/time impact
Efficiency	Late delivery of software	Computer time costs
	Feasibility of requirements	Staffing to conduct analysis
	Feasibility of design	Recruiting and training costs and times
		Added time and cost for review preparation
Flexibility	Lack of management visibility	Added time and cost to prepare inputs for reports
	Lack of a test discipline	Cost and time of using test group

Flexibility
Software organizations are driven by fast changing demands on skills and sheer numbers of engineers. With the development of a new and innovative product, many people are needed with broad experiences. However, when arriving in maintenance, these skill needs look different and manpower distributions are also changing. Such flexible demand cannot anymore be handled inside the enterprise. GSD is the answer to provide skilled engineers just in time and thus allows building flexible ecosystems combining suppliers, customers with engineering and service providers. Directly related risks to the flexibility goal are poor management visibility and distance and culture clashes (refer to Table 2.4).

2.7 Conclusion

There is a strong surge for global software development to countries with lower labor cost. This chapter promotes analysis of project drivers to gain insights into comparing development costs and time for distributed software development projects as compared to collocated projects.

Even though most of the evaluated software effort estimation techniques do not have any of the GSD-related cost/time factors included by default, these techniques are still suitable and applicable for estimation of GSD project with some setup and calibration work. Estimation methods such as estimation by analogy and algorithmic models can be applied to the development of distributed software if the person doing the estimation model setup is experienced in outsourcing. Then, the person would be able to include all necessary cost/time factors into the estimation model.

Also, all expertise-based techniques can be directly applied for GSD projects, but they require experts with experience and knowledge on GSD. The available development of distributed software specific techniques can naturally be also directly applied for GSD projects.

Future work of this research includes on one hand the verification and improvement of the factors of a distributed development project and on the other hand the application of methods on projects while collecting effort data to calibrate the relevance of each project driver.

References

1. Kile JF (2005) The Importance of effective requirements management in offshore software development projects. Doctoral dissertation, Pace University
2. Nguyen V, Steece B, Boehm B (2008) A constrained regression technique for COCOMO calibration. In: Proceedings of the second ACM-IEEE international symposium on empirical software engineering and measurement. ACM, pp 213–222
3. Anderson SD, Molenaar KR, Schexnayder CJ (2007) Guidance for cost estimation and management for highway projects during planning, programming, and preconstruction, vol 574. Transportation Research Board, Washington, DC
4. Wallace L, Keil M (2004) Software project risks and their effect on outcomes. Commun ACM 47(4):68–73
5. Ashiegbu BC, Ahaiwe J (2011) Software cost drivers and cost estimation in Nigeria. Interdiscip J Contemp Res Bus 3(8):431
6. Herbsleb JD (2007) Global software engineering: the future of socio-technical coordination. In 2007 future of software engineering. IEEE Computer Society, pp 188–198
7. Nicholson B, Sahay S (2001) Some political and cultural issues in the globalisation of software development: case experience from Britain and India. Inf Organ 11(1):25–43
8. Gopal A, Gosain S (2010) Research note-the role of organizational controls and boundary spanning in software development outsourcing: implications for project performance. Inf Syst Res 21(4):960–982
9. Holmstrom H, Conchúir EÓ, Agerfalk J, Fitzgerald B (2006) Global software development. challenges: a case study on temporal, geographical and socio-cultural distance. In: 2006 I.E. international conference on global software engineering (ICGSE'06). IEEE, pp 3–11
10. Lanubile F, Damian D, Oppenheimer HL (2003) Global software development: technical, organizational, and social challenges. ACM SIGSOFT Softw Eng Notes 28(6):2–2
11. Herbsleb JD, Moitra D (2001) Global software development. IEEE Softw 18(2):16–20
12. Nguyen T, Wolf T, Damian D (2008) Global software development and delay: does distance still matter? In: 2008 I.E. international conference on global software engineering. IEEE, pp 45–54
13. Betz S, Mäkiö J (2008) Amplification of the COCOMO II regarding offshore software projects. Offshoring of software development: methods and tools for risk management; [OUTSHORE; Proceedings], 33
14. Peixoto CEL, Audy JLN, Prikladnicki R (2010) Effort estimation in global software development projects: preliminary results from a survey. In: 2010 5th IEEE international conference on global software engineering. IEEE, pp 123–127
15. Teasley SD, Covi LA, Krishnan MS, Olson JS (2002) Rapid software development through team collocation. IEEE Trans Softw Eng 28(7):671–683
16. Yadav MS, Prabhu JC, Chandy RK (2007) Managing the future: CEO attention and innovation outcomes. J Mark 71(4):84–101
17. Noll J, Beecham S, Richardson I (2010) Global software development. And collaboration: barriers and solutions. ACM Inroads 1(3):66–78
18. Yadav V (2016) A flexible management approach for globally distributed software projects. Glob J Flex Syst Manag 17(1):29–40
19. Agerfalk PJ, Fitzgerald B, Holmstrom Olsson H, Lings B, Lundell B, Ó Conchúir E (2005) A framework for considering opportunities and threats in distributed software development
20. Smith PG, Blanck EL (2002) From experience: leading dispersed teams. J Prod Innov Manag 19(4):294–304
21. Kraut RE, Streeter LA (1995) Coordination in software development. Commun ACM 38 (3):69–82
22. Perry DE, Staudenmayer NA, Votta LG (1994) People, organizations, and process improvement. IEEE Softw 11(4):36–45

23. Sarker S, Sahay S (2004) Implications of space and time for distributed work: an interpretive study of US–Norwegian systems development teams. Eur J Inf Syst 13(1):3–20
24. Espinosa JA, Nan N, Carmel E (2007) Do gradations of time zone separation make a difference in performance? A first laboratory study. In: ICGSE, pp 12–22
25. Herbsleb JD, Grinter RE (1999) Splitting the organization and integrating the code: Conway's law revisited. In: Proceedings of the 21st international conference on Software engineering. ACM, pp 85–95
26. Niinimaki T, Piri A, Lassenius C (2009) Factors affecting audio and text-based communication media choice in global software development projects. In: 2009 fourth IEEE international conference on global software engineering. IEEE, pp 153–162
27. Casey V, Richardson I (2006) Uncovering the reality within virtual software teams. In: Proceedings of the 2006 international workshop on Global software development for the practitioner. ACM, pp 66–72
28. Lutz B (2009) Linguistic challenges in global software development: lessons learned in an international SW development division. In: 2009 fourth IEEE international conference on global software engineering. IEEE, pp 249–253
29. Lings B, Lundell B, Agerfalk J, Fitzgerald B (2007) A reference model for successful distributed development of software systems. In: International conference on global software engineering (ICGSE 2007). IEEE, pp 130–139
30. Herbsleb JD, Paulish DJ, Bass M (2005) Global software development at siemens: experience from nine projects. In: Proceedings of the 27th international conference on software engineering. ICSE 2005. IEEE, pp 524–533
31. Casey V, Richardson I (2008) The impact of fear on the operation of virtual teams. In: 2008 I. E. international conference on global software engineering. IEEE, pp 163–172
32. Carmel E, Agarwal R (2001) Tactical approaches for alleviating distance in global software development. IEEE Softw 18(2):22–29
33. PMBoK, A (2000) Guide to the project management body of knowledge. Project Management Institute, Pennsylvania
34. Lum K, Bramble M, Hihn J, Hackney J, Khorrami M, Monson E (2003) Handbook for software cost estimation. NASA Jet Propuls Lab JPL D-26303
35. Shepperd M, MacDonell S (2012) Evaluating prediction systems in software project estimation. Inf Softw Technol 54(8):820–827
36. Myrtveit I, Stensrud E, Shepperd M (2005) Reliability and validity in comparative studies of software prediction models. IEEE Trans Softw Eng 31(5):380–391
37. Miyazaki Y, Takanou A, Nozaki H, Nakagawa N, Okada K (1991) Method to estimate parameter values in software prediction models. Inf Softw Technol 33(3):239–243
38. Idri A, azzahra Amazal F, Abran A (2015) Analogy-based software development effort estimation: a systematic mapping and review. Inf Softw Technol 58:206–230
39. Jorgensen M, Shepperd M (2007) A systematic review of software development cost estimation studies. IEEE Trans Softw Eng 33(1):33–53
40. Wen J, Li S, Lin Z, Hu Y, Huang C (2012) Systematic literature review of machine learning based software development effort estimation models. Inf Softw Technol 54(1):41–59
41. El Bajta M, Idri A, Fernández-Alemán JL, Ros JN, Toval A (2015) Software cost estimation for global software development a systematic map and review study. In: Evaluation of Novel Approaches to Software Engineering (ENASE), 2015 international conference on. IEEE, pp 197–206
42. El Bajta M (2015) Analogy-based software development effort estimation in global software development. In: 2015 I.E. 10th international conference on global software engineering workshops. IEEE, pp 51–54
43. Muhairat M, Aldaajeh S, Al-Qutaish RE (2010) The impact of global software development factors on effort estimation methods. Eur J Sci Res 46(2):221–232
44. Jorgensen M (1995) Experience with the accuracy of software maintenance task effort prediction models. IEEE Trans Softw Eng 21(8):674–681

45. Amazal FA, Idri A, Abran A (2014) Software development effort estimation using classical and fuzzy analogy: a cross-validation comparative study. Int J Comput Intell Appl 13 (3):1450013
46. Jain MS (2012) Survey of various cost estimation techniques. Int J Adv Res Comput Eng Technol (IJARCET) 1(7):229
47. Boehm BW, Madachy R, Steece B (2000) Software cost estimation with Cocomo II with Cdrom. Prentice Hall PTR, Upper Saddle River
48. Boehm B, Clark B, Horowitz E, Westland C, Madachy R, Selby R (1995) Cost models for future software life cycle processes: COCOMO 2.0. Ann Softw Eng 1(1):57–94
49. Kemerer CF (1987) An empirical validation of software cost estimation models. Commun ACM 30(5):416–429
50. Sosa ME, Eppinger SD, Rowles CM (2004) The misalignment of product architecture and organizational structure in complex product development. Manag Sci 50(12):1674–1689
51. O'Hara M, Johansen R (1994) Global work: bridging distance, culture and time. Jossey-Bass, San Francisco
52. Sosa ME, Eppinger SD, Pich M, McKendrick DG, Stout SK (2002) Factors that influence technical communication in distributed product development: an empirical study in the telecommunications industry. IEEE Trans Eng Manag 49(1):45–58
53. Boehm BW (1988) A spiral model of software development and enhancement. Computer 21 (5):61–72
54. Tufekci O, Cetin S, Arifoglu A (2010). Proposing a federated approach to global software development. In Digital Society, 2010. ICDS'10. Fourth International Conference on. IEEE, pp 150–157
55. Ebben JJ, Johnson AC (2005) Efficiency, flexibility, or both? Evidence linking strategy to performance in small firms. Strateg Manag J 26(13):1249–1259