Modelling software development across time zones

Article *in* Information and Software Technology · January 2006

DOI: 10.1016/j.infsof.2004.02.006 · Source: DBLP

2 authors:

Adel Taweel
Birzeit University
89 PUBLICATIONS **429** CITATIONS
SEE PROFILE

Pearl Brereton
Keele University
169 PUBLICATIONS **3,752** CITATIONS
SEE PROFILE

Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information and Software Technology 48 (2006) 1–11
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Modelling software development across time zones

Adel Taweel[a], Pearl Brereton[b,*]

[a]School of Computer Science, University of Manchester, Kilburn Building, Manchester M13 9PL, UK
[b]School of Computing and Mathematics, Keele University, Keele, Staffordshire ST5 5BG, UK

Received 31 January 2003; revised 20 February 2004; accepted 23 February 2004
Available online 10 May 2005

Abstract

Economic factors and the World Wide Web are turning software usage and its development into global activities. Many benefits accrue from global development not least from the opportunity to reduce time-to-market through 'around the clock' working.

This paper identified some of the factors and constraints that influence time-to-market when software is developed across time zones. It describes a model of the relationships between development time and the factors and overheads associated with such a pattern of work. The paper also reports on a small-scale empirical study of software development across time zones and presents some lessons learned and conclusions drawn from the theoretical and empirical work carried out.

Keywords: Global software development; Development across time zones; Distributed software engineering

1. Introduction

Market forces and an increasingly reliable world-wide communications network have made geographically distributed software development a reality. The potential benefits of global software development have been well documented and include [1–3]:

- Improved product and service quality;
- Rapid response to global market opportunities;
- Reduced time-to-market through round the clock working;
- Reduced costs;
- Better use of scarce resources;
- Adding an international perspective to products developed.

On the negative side, it has been reported that multi-site software development introduces additional delay and thus takes much longer than single-site development [4]. This delay is attributed to the fact that more people are normally involved in multi-site software development (compared to development on a single site), however, the projects studied do not seem to have utilised time difference between sites and thus have not exploited an 'around the clock' work pattern.

In this paper, we particularly focus on the opportunities to reduce time-to-market through 'around the clock' software development i.e. through the exploitation of time differences between development sites. The working style that we are interested in is where a task is passed at the end of a working day from one software engineer to another 'across time zones'. We call this *sequential collaborative software engineering* (SCSE).

Working around the clock is not, of course, a new idea. In fact, in some domains such as healthcare and air traffic control, shift working is common practice. With software, however, the benefits of around the clock working can be achieved though *work transfer across time zones*, thereby eliminating the need for unsociable working hours.

The aims of the research reported here are to gain a better understanding of the *contextual factors* that affect time-to-market for sequential collaborative software engineering and to investigate the *overheads* associated with such work patterns.

In particular, a model, *Tseq* (Time Estimation for Sequential Collaborative Software Engineering)

* Corresponding author. Tel.: +44 178 258 3079; fax: +44 178 271 3082.
E-mail addresses: a.taweel@cs.man.ac.uk (A. Taweel), o.p.brereton@keele.ac.uk (P. Brereton).

0950-5849/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2004.02.006

representing the relationships between the factors, the overheads and time-to market is described. Such a model could be used to help project managers decide when and in what manner software engineering activities should be distributed across time zones (assuming that the necessary expertise is available at the participating sites) in order to achieve significant reductions in completion time.

Also briefly described in the paper, is a small-scale empirical study undertaken to illustrate the feasibility of SCSE and to obtain some practical measures for the contextual factors and overheads of distribution.

Section 2 of the paper describes and illustrates patterns of distributed working. Section 3 identifies and discusses the contextual factors while Section 4 addresses the overheads of development across time zones. Section 5 introduces *Tseq* and discusses the affects on development time of some of the factors and overheads. Section 6 describes the empirical study carried out. Finally, Section 7 presents some conclusions.

2. Patterns of working across time zones

Working around the clock is normally done by teams of people where one team transfers work to another team. A team might simply take over the tasks of the previous team or it might undertake some additional transfer tasks. For example, when security officers take over from their colleagues, they may be required to do certain checking activities before they resume their normal task. In most domains, however, a team member continues to work from the point where the previous person finished on the same or similar type of job. For example, nurses continue from the point where colleagues on a previous shift stopped, effectively doing the same type of job.

This concept can be utilised for software engineering tasks. Fig. 1 shows a general 3-site scenario for around the clock software development. The time needed to report progress at the end of a day, and to catch up at the beginning of the day (*reporting time* and *catching-up time*, respectively) depend on many factors. These include, for example, the complexity of the task and the opportunity for synchronous communication (i.e. the length of the time overlap).

For the purpose of our model, we consider two ways of distributing software engineering tasks across time zones.

1. *Sequential task distribution*. In this case, a task, which is normally done by one person, is split between two (or more) persons located in different time zones. That is, one person transfers a task to a second person located in a different time zone, and the second person continues the work from the point where the first person has stopped. With this process, more than one person-day can be fitted into a single day (24-h period).
2. *Dependent task distribution*. In this case, one task is done by one person located in a particular time zone and a dependent task is subsequently undertaken by another person located in a different time zone. For example, the tasks might be coding and testing. One person, located in the UK, can do coding and then transfer the code (a testable part) to another person located in the USA to test it. Another example is writing a test plan and executing it. These tasks may be considered sequential, however, the different individual tasks have to be executed in a certain sequence as opposed to being considered as one sequential task made up of smaller activities that can be executed in any order.

3. Contextual factors affecting development time

A number of factors are likely to affect time-to-completion for SCSE. In particular, we identify factors relating to the development sites used, the project resources

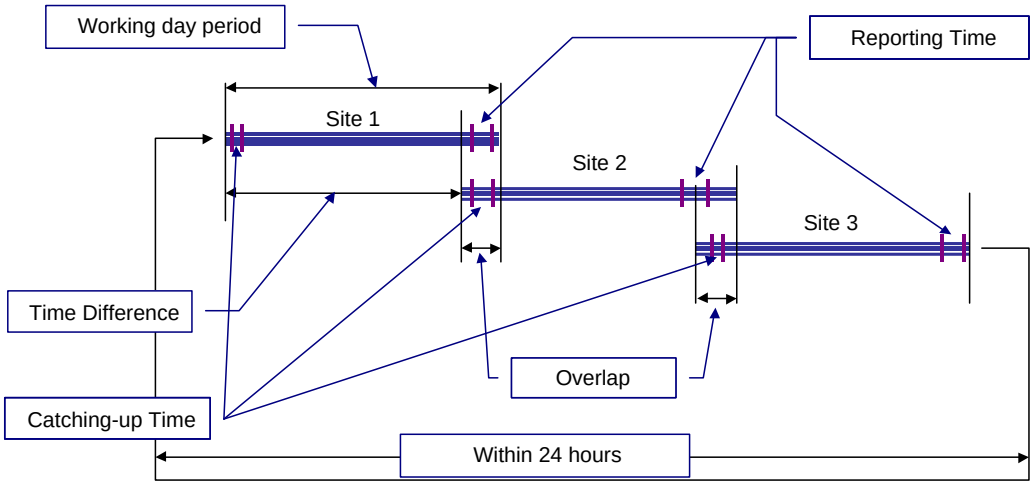

Fig. 1. Three sites.

and constraints and the characteristics of the task being undertaken.

3.1. Characteristics of the development sites

Many international organisations have a number of sites located in different countries around the world. Other smaller organisations could form an international virtual organisation from business partners located in various countries. In either case, these sites can be utilised to support SCSE. The potential for reducing development time will depend on the number of sites involved and on the time differences between sites. The maximum number of sites that can be utilised for one task depends on the length of the working day at each site and on the time difference between sites.

For SCSE to succeed, participating sites must have sufficient communications facilities to support the reliable transfer of software and progress information between sites. Information exchange can be carried out in a number of ways including through the use of a central repository, using asynchronous communications (e.g. email or ftp), using synchronous communications (e.g. video conferencing) or through some combination of these.

3.2. Project resources and constraints

Clearly, the development time for a particular software system will depend on the number of developers employed. However, the optimal number of developers that can be utilised will depend on the characteristics of the activities involved. For example, it will depend on the number of possible parallel activities that can be carried out autonomously. Brooks [5] noted that communication overheads would increase by $n(n-1)/2$ for n developers. However, for SCSE (involving the work patterns described in Section 2), most of the communication overhead is likely to occur when work is 'handed over' from one developer to another at the end (and beginning) of a shift. At other times, the co-developers will not be available unless there is a significant overlap in the working days. Communication is, therefore, mainly between only the two sites involved in work transfer and so the overhead is likely to be less. Rather it might be expected to increase linearly with the number of sites.

The usual project constraints such as delivery date and maximum cost are also likely to affect development time.

3.3. Task characteristics

These include:

Estimated single-site development time. The estimated time to completion for development across time zones will of course be a function of the estimated development time if the task were to be carried out at a single site (which reflects the overall size and complexity of the task being undertaken).

Level of concurrency and critical path. The potential for concurrent working, which will depend on the nature of the task, will clearly affect development time.

The critical path within a project determines the shortest possible time a project or a set of tasks can take to complete, if concurrency is exploited to the maximum. Therefore, critical path/number of sites is the shortest possible time to finish a project or a set of tasks. Consequently, the reduction in development time will have a value between (total effort/number of sites + overheads) and (critical path/number of sites + overheads).

4. Overheads of SCSE

The overheads of distribution can be classified into three main categories: management, knowledge transfer and distribution effort loss.

4.1. Management overhead

The added complexity of SCSE over single-site development is likely to result in a need for more planning and monitoring of progress. Extra planning will be needed for actually deciding which tasks are to be distributed across which sites and, because more work will be done in a shorter period, further effort is likely to be required to monitor progress [6,7]. Estimating the values of these overheads is not straightforward, as they will depend (non-linearly) on project characteristics such as overall size and schedule. Estimates for these overheads are probably best determined from historical data within a particular organisation.

4.2. Knowledge transfer overhead

Knowledge transfer overheads include *general task-level communications*, *artefacts transfer time* and *daily knowledge transfer*. General task-level communications may be needed from time to time, for example, to negotiate the overall approach, to review the decision making process or to clarify any decisions made.

General task-level communication can be carried out using either synchronous, asynchronous communication or both between distributed team members. While asynchronous communication can happen spontaneously based on events and requirements during a task's or project's life, synchronous communication needs pre-arranging or pre-scheduling and probably some preparation to succeed which may incur additional overheads. However, because distributed team members will be located in different time zones, the use of synchronous communication will be

limited. Although, it is apparent that longer periods of overlap allow more sites to be utilised within a 24-h period and increase the opportunities of synchronous communication, they are not necessarily an advantage. Longer periods of overlap between sites allow less utilisation of a person's day which reduces a person's working time on tasks or projects. It also contributes to other overheads such as artefacts transfer time and catching-up time overheads (due to the possible increase in the number of participating sites) and general task-level communication overhead (due to the possible increase in periods of synchronous interactions). Therefore, the overlap between two sites, for at least one site will be a wasted time for the task being carried out, although if it is long enough it can be used to carry out some other tasks. Therefore, in most cases, the use of asynchronous communication with SCSE contributes more toward improving time-to-market. Although, the importance of the availability of synchronous communication is emphasised [8,9], the use of asynchronous communication is considered to be sufficient for many software tasks [10,11]. It also helps to overcome cultural and language barriers and decreases communication overheads due to face-to-face interactions [10–14].

Several researchers have noted the importance and impact of co-operation and communication overheads on development time between team members [5,15]. However, in SCSE these overheads are not significant. Brooks noted that communication overheads would increase by $n(n\text{-}1)/2$ for n developers, thus for a large team the development time increases significantly [5]. Based on these overheads, Brooks draws his law: 'adding staff to a late project makes it later'. However, most of the communication overhead is likely to occur when work is 'handed over' from one developer to another at the end (and beginning) of a shift. Communication is, therefore, mainly between only the two sites involved in work transfer and so the overhead is likely to be less. Rather, in this case, it is expected to increase linearly with the number of sites. In addition, due to the significant time difference between sites, interaction between members located at these sites will be limited.

General task-level communication should not be confused with the time needed to transfer software artefacts (referred above as *artefacts transfer time*) which will occur on a daily basis. This overhead is not only limited to the time taken for artefacts to be transferred but also includes the time required to launch communication tools and prepare artefacts for transmission. As mentioned above, different communication methods can be used to transfer artefacts between sites, such as through the use of a central repository, distributed repositories, e-mail attachments or FTP, Internet, or dedicated private communication networks. The time needed to transfer these artefacts will depend on the transfer method used. The empirical study, described in Section 6, shows that this overhead depends on how well a site is equipped. Because this overhead depends on many factors such as the data transfer method, the size of data, the communication tools and methods and the users' familiarity with the technologies used, it is probably best determined from historical data within a particular site and organisation.

In addition, for both the *sequential* and *dependent* distribution patterns, *daily knowledge transfer* between sites is needed. This consists of two elements:

Reporting time. At the end of a day, the time taken to report progress will depend on the number of activities/tasks to be reported, the quantity of knowledge to be reported on each task, clarity of project schedule/plan and on the level of automation of the reporting process.
Catching-up time. At the beginning of a day, the time needed to 'catch up' on the work carried out since a developer's last working day will depend on a number of factors. These include: how well the transferred knowledge is presented; the number of activities/tasks reported; the clarity of project schedule/plan and the level of understanding of the whole task structure or plan. In addition, catching-up time depends on the number of sites involved with the task. This is because a person at a site may need to catch-up with (number of sites − 1) status reports. Although some of the factors identified are easily quantified (e.g. number of activities reported) other are difficult to measure (e.g. the level of automation).

4.3. Distribution effort loss

Distribution effort loss represents the additional effort lost due to unfinished tasks [2]. This happens only with *dependent* type tasks. For example, consider the coding and testing example mentioned above. If one team at a site does not finish coding the expected part, the next person would not be able to test it. Absence, illness, technical problems etc. can prevent software engineers from completing a task.

This potential for effort loss is illustrated by an 2-site scenario where a developer at site A is working on task 1, a developer at site B is working on task 2 and task 1 is dependent on task 2 (see Fig. 2). It can be seen that 1 working day is lost at site A if the developer at site B fails to complete the required part of task 2 on day 2.

If the dependency level between tasks is very high, then effort loss may be high and, at worst, will be (number of sites-1) person days in every transaction (or 1 day of schedule time). If the dependency level between tasks is low, the loss is 'zero' person days.

However, if an uncompleted task can be completed by a different person the loss will always be 'zero'. The dependency/inter-dependency between tasks can be accurately determined only when tasks are scheduled.

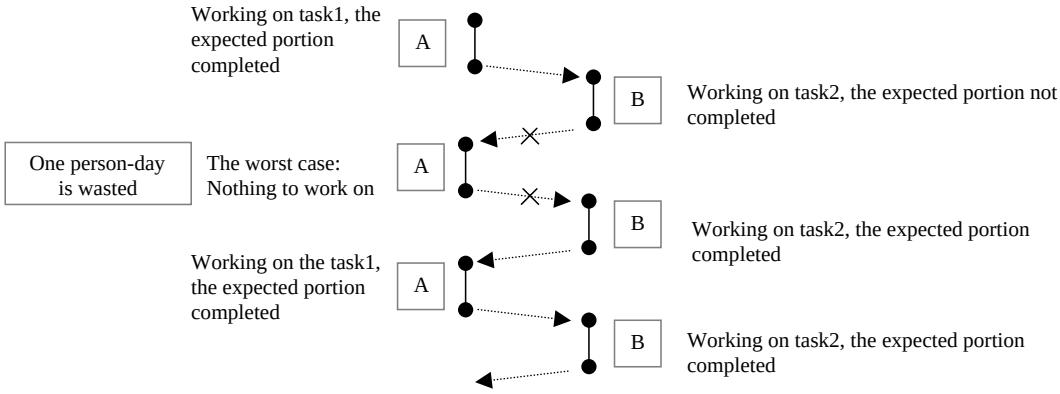

Fig. 2. Distribution effort loss.

5. The *Tseq* model

A mathematical model (*Tseq*) of the relationships between development time for SCSE and the contextual factors and overheads of distribution has been developed and is summarised below. The contextual factors represented in the model are:

- number of sites;
- time differences between sites;
- number of developers;
- estimated single-site development time;
- level of concurrency (i.e. the task specific potential for concurrent working).

The overheads of distribution included in the model are:

- management overhead;
- knowledge transfer overhead: general task-level communications (or collaboration time), artefact transfer time, reporting time, and catching up time;
- distribution effort loss.

The full derivation and discussion of the *Tseq* model is beyond the scope of this paper, however, a description of the main equations of the model and how they might be used are included. In deriving the *Tseq* model, the software process and development time of single site software development was taken as a baseline. The ratio of gain with respect to single site software development is, therefore, calculated and represented in two gain factors: a gain factor due to utilisation of more sites and a gain factor due to utilisation of more effort. These two gain factors (GF_{N_s} and GF_{N_e}) can be represented by Eqs. (1) and (2), respectively, where:

N_s is the number of utilised sites;
N_e is the number of developers;
O_i is the overlap between consecutive sites;
GF_{N_s} is the gain factor based on the number of utilised sites;
GF_{N_e} is the gain factor based on the number of developers;
WD is the working day at each site (it is assumed that the working day at all sites has the same value).

$$GF_{N_s} = \sum_{i=1}^{N_s-1} \left(\frac{WD - O_i}{N_s \times WD} \right) \quad (1)$$

$$GF_{N_e} = \sum_{i=1}^{N_e-1} \left(\frac{WD - O_{si}}{N_e \times WD} \right) \quad (2)$$

where O_{si} is the overlap between ith site where the ith developer is located and the $i+1$ site where $i+1$ developer (member of the team) is located. $O_{si}=0$ for developers who are not part of a team.

Consequently, the corresponding potential reduction in development time ratios (or Reduction Factors, RF_{N_s} and RF_{N_e}) can be expressed in the following Eqs. (3) and (4):

$$RF_{N_s} = 1 - GF_{N_s}, \quad 0 < RF_{N_s} \leq 1 \quad (3)$$

$$RF_{N_e} = 1 - GF_{N_e}, \quad 0 < RF_{N_e} \leq 1 \quad (4)$$

where $RF_{N_s} \geq RF_{N_e}$

For a given software task, some parts may need to be carried out sequentially whilst other parts may be carried out concurrently. The estimated development time for SCSE can be represented by Eq. (5), where:

ST is the development time estimate for sequential tasks;
PT is the development time estimate for parallel tasks;
EDT is the development time estimate for single site development;
PW is the level of potential for concurrent working in the given software task; and
NDT is the multi-site development time estimate.

$$NDT = ST \times RF_{N_s} + PT \times RF_{N_e} + Overheads \quad (5)$$

where $ST = EDT - EDT \times PW$, $PT = EDT \times PW$, and, $Overheads = KTO + DELO$ (see Eqs. (7) and (8) below).

The above equations are based on the assumption that all utilised sites work within a 24-h period. This can be

expressed by the following Eq. (6)

Total working period

$$= \sum_{i=1}^{N_s} \mathrm{WD}_i - \left(\sum_{i=1}^{N_s - 1} \mathrm{WD}_i - \mathrm{TD}_i \right) \quad (6)$$

The equations for each overhead are derived differently depending on the nature of the overhead. For example, due to the randomness in the occurrences of the *distribution effort loss* (DELO) overhead, it would not be appropriate to calculate it as a daily effort loss. It is more reasonable to consider it as a proportion of the total software task elapse time. Although, the proportion will differ from one project to another, a sensible value can be derived from the historical data of an organisation. This can be expressed as

$$\mathrm{DELO} = \mathrm{LP} \times \mathrm{EDT} \times (N_s - 1) \quad (7)$$

where EDT represents the multi-site development time, which in this case includes both NDT and other overheads, LP represents (as a percentage) the occurrences of Daily-Transfer effort loss with respect to the software task schedule time, and N_s is the number of utilised sites.

Knowledge transfer overhead contributes to the elapse time of a software task on a daily basis; for each transfer of each team on each site. The equations for the knowledge transfer overhead (KTO) are, therefore, derived for each site taking NDT as the baseline. To converge to the near true value of KTO, it has been calculated iteratively until the error is less than 1 day. Therefore, the knowledge transfer overhead can be expressed as:

$$\mathrm{KTO}_{\mathrm{Total}} = \sum_{i=1}^{n} \mathrm{KTO}_i \quad (8)$$

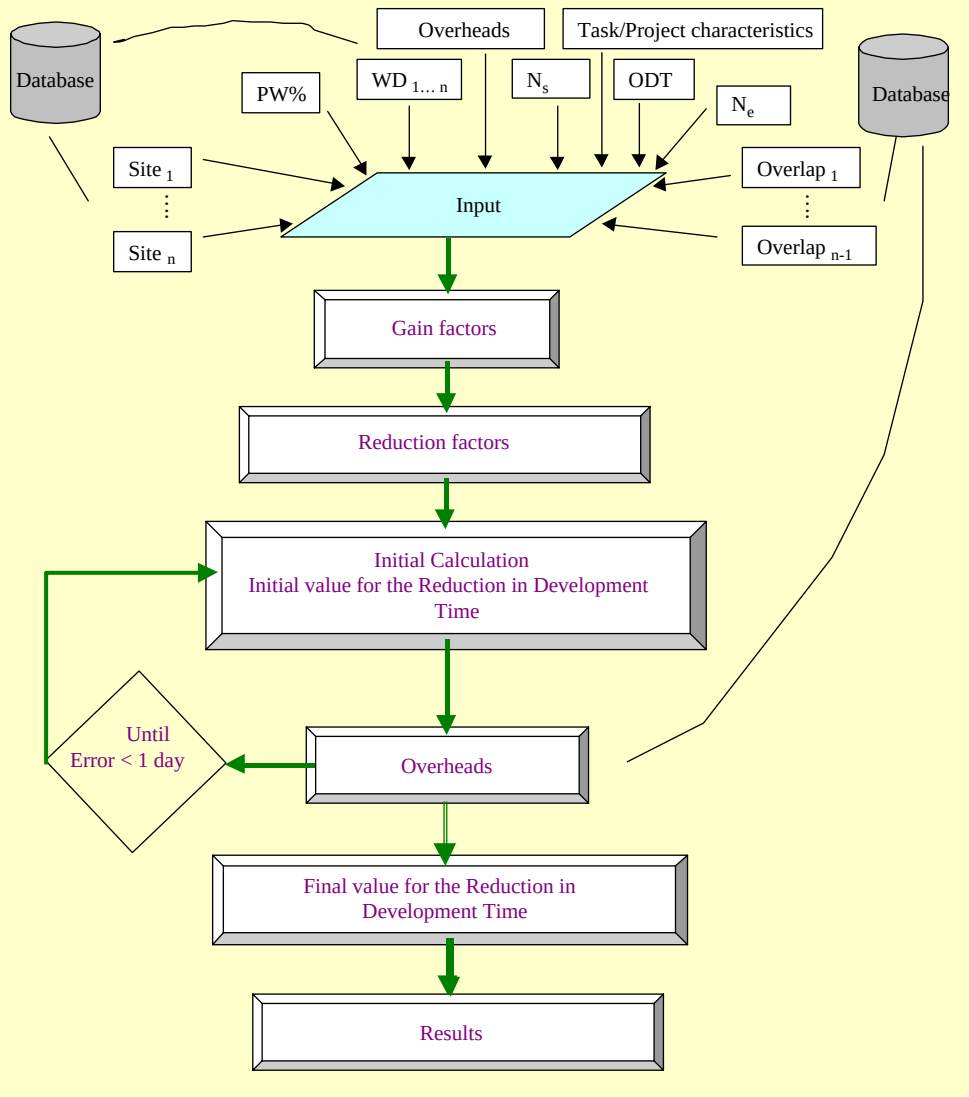

Fig. 3. Development time estimation process.

where n is largest number such that $KTO_n \geq 1$ day such that

$$KTO_1 = KTO_{LF} \times NDT_{init},$$

$$KTO_2 = KTO_{LF} \times KTO_1, \ldots, KTO_n = KTO_{LF} \times KTO_{n-1}$$

and KTO_{LF} (knowledge transfer overhead loss factor)

$$KTO_{LF} = \frac{GCoT_{day}}{N_s WD} + \frac{KTT_{day}}{N_s WD} + \frac{ATT_{day}}{N_s WD}$$

where, $GCoT_{day}$, KTT_{day} ATT_{day} are general task-level communication, knowledge transfer time (catching-up and reporting time) and artefact transfer time for all teams on all sites in a day (24 h) respectively. The calculation of ATT_{day} is slightly more complicated than $GCoT_{day}$ and KTT_{day}, because it depends not only on the method of transfer between sites, but also, on the number of teams, size of each team and location of team members. The equations for calculating ATT are not included.

An illustration of how the above equations might be used to estimate multi-site development time is shown in Fig. 3.

As stated in the introduction, our objectives are:

1. to gain a better understanding of the contextual factors of relevance to SCSE;
2. to investigate the overheads of SCSE;
3. to enable the development time to be estimated when SCSE is used.

In order to address the first two objectives, an implementation of *Tseq* has been used to determine the effects on development time of variations in values for particular factors and overheads.

So, for example, if we consider a project with an estimated single-site development time of 12 person months we can determine the effect on development time of using multiple sites. As might be expected, development time decreases as the number of sites increases (see Fig. 4). Of course, in practice, as the number of sites is increased the value of some contextual factors will change. For example, the time differences between sites will decrease and some

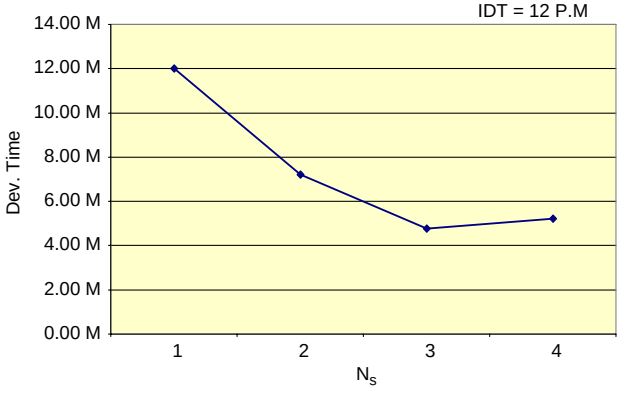

Fig. 4. Development time vs. number of sites (N_s).

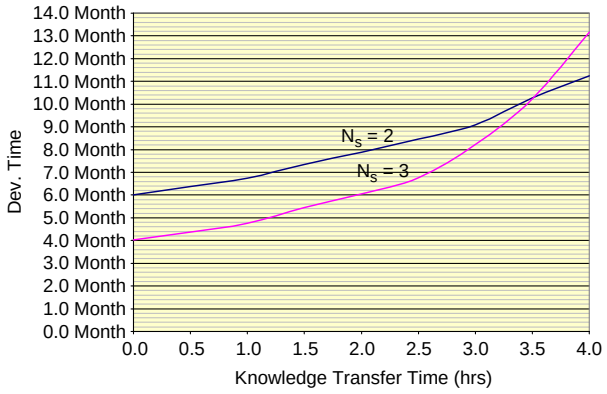

Fig. 5. Development time vs. knowledge transfer time.

overheads such as daily knowledge transfer time may increase due to the need to catch up with work from more sites (or it may decrease because less work is done during each shift). In practice, utilising more than three sites is unlikely to reduce development time.

A more interesting case is to consider the effect of daily knowledge transfer time on development time. Fig. 5 shows this (non-linear) relationship for both a 2-site and a 3-site distribution. Here, it can be seen that if the daily knowledge transfer time exceeds 3.5 h then it would be faster to use two sites rather than three.

In order to address objective 3, an algorithm based on the *Tseq* model, and embodied in a decision support tool, could

Typical Input

EDT	Estimated single site development time
N_s	Maximum number of sites available
TD_i	time difference between sites
RT	Reporting time
CT	Catching-up time
N_e	Number of developers
PW	Level of concurrency
GCO	General task-level communications

Processing

1. Calculate gain factors and reduction factors
2. Calculate initial development time (for distributed working)
3. Calculate overheads
4. Recalculate development time
5. Repeat step 4 and step 5 until error < 1 day
6. Recalculate development time with respect to overheads

Typical output

Estimated development time for distributed working using all available sites
Estimated development time using < all sites
Expected values of overheads

Fig. 6. The *Tseq* algorithm.

provide estimates of development time for (say) 2-site and 3-site distribution together with the total overhead costs. The accuracy of the estimates will, of course, depend on the quality and range of input values. Typical input and output parameters together with the processing involved are shown in Fig. 6. A prototype tool has been developed and is currently being evaluated.

6. The empirical study

In order to show the feasibility of SCSE and to obtain some real measures for contextual factors and overheads, an exploratory empirical study was carried out using software engineers (subjects) at two sites (Keele and Hebron) to undertake a software development task (details of the study are published in [16]). Although, the time zones for these sites are only 2 h apart, they were considered suitable for the following reasons:

- The computing facilities at each site were similar.
- Subjects with similar cultural backgrounds and with good English language skills were available at both sites. Cultural and language differences are of course important issues in global software engineering and are the focus of a number of other studies [17,18].
- Available subjects were mature, enthusiastic and had very good (and comparable) software engineering skills.

The four subjects were male, aged between 29 and 39 years with between 6 and 11 years programming experience and between 10 months and 3 years Java programming experience. The subjects were all familiar with the development method notations and coding conventions used and had an established level of trust between them. The importance of such trust between team members has already been noted by several researchers and can often determine the success or failure of teams and projects [19–21].

Two sets of two subjects worked collaboratively, over five shifts, each of approximately 3 h duration, on the given software task. The software task (which is summarised in the Appendix A) included three main activities: implementation, unit-testing and application testing. The implementation and unit-testing activities were carried out in each shift, however, application testing was carried out only in the last two shifts. Both quantitative and qualitative data was collected. Quantitative data included time spent on the task for each shift, catching up time, reporting time and time spent communicating. Average values for each are shown in Table 1. The variations in values for catching up time and reporting time were small, however, the communications times for Hebron shifts were considerably longer than for the Keele shifts. This was because the Hebron subjects had to use a dial-up connection whereas Keele subjects had a permanent Internet connection. As discussed in Section 3.1, it is clearly important that the participating sites have adequate communications facilities to support SCSE.

Table 1
Quantitative data from the empirical study

	Average (in minutes)
Development time	98
Catching up time	5
Reporting time	8
Communications time	18

Table 2
Summary of questionnaire responses

	Yes	No
Was synchronous communications needed?	0	4
Were design documents adequate?	4	0
Were problems encountered with the communications?	1	3
Do you think that SCSE would be more suitable for bigger projects?	2	1 (+1 unsure)
Did you feel that SCSE restricted your work?	3	1
Would you prefer to use SCSE as opposed to single-site development?	0	2 (+2 unsure)

The quite low values for catching up time and reporting time are encouraging especially given the low level of automation available to the subjects in the study to support these activities.

Qualitative data was concerned with the subjects' opinions on a number of issues. Some of these are included in Table 2. Although, from their general comments, the subjects felt SCSE to be very efficient it was not a particularly popular way of working. This seems to be because, to some extent, they felt that their working style was restricted. Nevertheless some were keen to try the approach for larger scale projects.

Of the general comments from subjects, the most notable was an expression of the need for high quality documentation especially relating to requirements, design, design rationale and coding conventions.

One of the lessons learned from the study was that it is important (and difficult) to plan to the right level of detail. If subjects finished their allocated activities early they did not continue to work until the end of the shift (even though they had been told to do so). However, subjects also felt that they were 'under pressure' to complete the activities scheduled. Clear a balance needs to be struck between the view that 'I have done my bit so I can stop' and 'I can't keep up with the work rate expected of me'.

7. Conclusions

This paper has investigated some of the issues that arise when software is developed across time zones.

In particular, some of the contextual factors relating to participating sites and to project and task characteristics that support or constrain distributed working have been highlighted. These factors, together with the overheads of distribution have been combined into a set of equations which can be used to estimate development time for sequential collaborative software engineering.

As well as these theoretical investigations, a small-scale empirical study of SCSE is also reported.

A number of conclusions arise from this work. In particular, it is clear that documentation is a crucial factor. In fact, as well as being a necessity for successful SCSE, the data recorded for the purposes of knowledge transfer is a valuable contribution to the documentation process.

Other benefits of SCSE, in terms of quality and maintainability arise from:

- more than one person being involved in development (thereby achieving some of the benefits of pair programming [22]);
- a well-documented process;
- adherence to coding conventions.

To make SCSE a more controlled and predictable process there is a need for:

- greater automation of the knowledge transfer process;
- better ways of monitoring progress;
- use of a central repository and good communications to support knowledge management;
- further research to investigate the level of detail needed for scheduling of activities between collaborators.

Acknowledgements

The authors acknowledge the support of British Telecommunications plc who funded this work and the subjects who took part in the empirical study.

Appendix A. Task requirements specification and implementation schedule

System Requirements

System Name: Calculator

Requirement Specifications:

This application is a normal common calculator. It should have the following specifications:

1. An applet based application that runs in a Java-based browser
2. Can be activated using the mouse.
3. Performs the following mathematical functions
   - 3.1. Addition (+)
   - 3.2. Subtraction (-)
   - 3.3. Multiplication (*)
   - 3.4. Division (/)
   - 3.5. Power (x^y)
   - 3.6. Square (x*x)
   - 3.7. Percent (%)
   - 3.8. Square Root
4. It has a standard interface

Exclusions:

1. It does not provide a help systems
2. It does not include any menu/options for upgrade.
3. It runs on a browser that supports the current version of Java (version 1.2)
4. It can only be activated using the mouse and not the keyboard

Behavioural Restrictions:

1. Display: Numbers should be left aligned with respect to the display. Calculator when started/restarted/reinitialised should display 0.0. Numbers when displayed should be in the double format (e.g. 11.0, 11.2, 0.112 etc.)
2. One Operand Functions: For these functions enter a number and then requesting a function should do the calculation on the entered number and display the result.
3. Two operand Functions: for these functions enter a number, request a function, enter another number and then requesting another function or get a result function (Equal) displays the result of the first requested function.
4. Get a result function (Equal): execute the last requested operation (for two operand function only) and displays the result of the calculation. Subsequent request of this function should not invoke any function or change displayed result

Implementation Schedule

	Hebron		Keele	
Shift 1	***Class***	***Components***		
	Calculator	init() – the interface only		
	SysAction	All		
	Display	All		
Shift 2			***Class***	**Components**
			Accumulator	All
			Operation	All
			Math Functions	Add(), Sub(), Div() Power()
Shift 3	**Class**	**Components**		
	Math Functions	Mult(), Square(), Percent() SqRoot()		
	Control Variables	All		
Shift 4			**Class**	**Components**
			EventHandler	ClearButton() NumbersButton() DecimalPointButton()
Shift 5	***Class***	**Components**		
	EventHandler	ExecuteOperation() OneOperandOperationButton() TwoOperandOperationButton EqualButton()		
	SysAction	actionPerformed() – Update		
Shift 6			**Task**	**Components**
			Integration and Testing	All

References

[1] F. Maurer, Computer support in project coordination, Proceedings of the WET ICE'96, California, USA, IEEE Computer Society, Fifth International Conference on Enabling Technologies, June 1996.

[2] I. Gorton, S. Motwani, Issues in co-operative software engineering using globally distributed teams, Inform. Software Technol. 38 (1996) 647–655.

[3] E. Carmel, Global Software Teams: Collaborating Across Borders and Time Zones, Prentice Hall, Englewood Cliffs, NJ, 1999.

[4] J.D. Herbsleb, R.E. Grinter, T.A. Finholt, An empirical study of global software development: distance and speed, Proceedings of the ICSE'2001, Toronto, Canada, May 2001, pp. 81–90.

[5] F.P. Brooks, The Mythical Man-Month: Essays on Software Engineering, Addison-Wesley Longman Inc, Reading, MA, 1995. Anniversary Edition.

[6] R. Fournier, Teamwork is the Key to Remote Development, Infoworld, San Francisco, USA, March, 2001.

[7] G. Anthes, Divide and Conquer, Computerworld, Framingham, MA, June 2000.

[8] R. Daft, R.H. Lengel, Information richness: a new approach to managerial behavior and organization design in: B. Staw, L. Cummings (Eds.),, Research in Organizational Behavior vol. 6, JAI Press Inc, Greenwich, CT, 1984, pp. 191–233.

[9] N. Nohria, R.G. Eccles, Face-to-face: making network organizations work, in: Nohria/Eccles (Hrsg.), Networks and Organizations Structure, Forms and Action, Harvard Business School Press, Boston, 1992, pp. 188–308.

[10] C. Maitland, C. Steinfield, C.Y. Jang, Supporting globally distributed engineering design teams with communication technologies, Proceedings of the Joint Conference, Calgary, Canada, Global networking, vol. 2, IOS Press, Amesterdam, 1997. pp. 338–346.

[11] A. French, P. Layzell, A study of communication and cooperation in distributed software project teams, Proceedings of the International Conference on Software Maintenance, Bethesda, Maryland, 1998 pp. 146–155.

[12] K. Hussein, F. Pena-Mora, R.D. Sriram, CAIRO: A System for Facilitating Communication in a Distributed Collaborative Engineering Environment, Proceedings of the WET ICE'95, IEEE Computer Society Press, 1995. pp. 154–162.

[13] C.B. Seaman, V.R. Basili, Communication and organization in software development: an empirical study, IBM Syst. J. 36 (4) (1997) 550–564.
[14] A. Van der Smagt, Enhancing virtual teams: relations vs. communication technology, Proceedings of the Fourth American Conference on Information Systems, Baltimore, Maryland, 1998 pp. 574–577.
[15] I. Sommerville, Software Engineering, fifth ed., Addison-Wesley Publishing Company Inc, 1995. ISBN-0-201-42765-6.
[16] A. Taweel, O.P. Brereton, Developing software across time zone: an exploratory empirical study, Informatica 26 (3) (2002) 333–344 ISSN 0350-5596.
[17] J. Lipnack, J. Stamps, Virtual Teams: Reaching Across Space, Time and Organizations with Technology, Wiley, New York, 1997.
[18] F. Lau, S. Sarker, S. Sahay, On managing virtual team, Technical Report Series 1999 No. 1, University of Alberta, Edmonton, Alberta, Canada, 1999.
[19] D.E. Perry, G.S. Gil, Votta, G. Lawrence, A case study of successful geographically separated teamwork, Proceedings of the SPI98, 1998.
[20] P. Brereton, S. Lees, M. Gumbley, C. Boldyreff, S. Drummond, P. Layzell, L. Macaulay, R. Young, Distributed group working in software engineering education, Inform. Software Technol. 40 (1998) 221–227.
[21] T. Ishaya, L. Macaulay, The role of trust in virtual teams, Proceedings of the Second International VoNet–Workshop Zurich, Verlag, Berlin, 1999. pp. 140–160.
[22] L. Williams, R.R. Kessler, W. Cunningham, R. Jeffries, Strengthening the case for pair-programming, IEEE Software 2000;.