Developing Software Across Time Zones: An Exploratory Empirical Study.

Article *in* Informatica · November 2002

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

Developing Software Across Time Zones: An Exploratory Empirical Study

Adel Taweel and Pearl Brereton
Department of Computer Science
Keele University, Keele
Staffordshire ST5 5BG, UK
Email: a.taweel or o.p.brereton@cs.keele.ac.uk
www.keele.ac.uk/depts/cs

Keywords: global software development, collaborative working, distributed software engineering, virtual teams.

Received: May 11, 2002

Market forces and an increasingly reliable world-wide communications network have made geographically distributed software engineering a reality. Global software development enables businesses to respond more easily and more quickly to global market opportunities and to improve product and service quality. One of the many potential benefits of global development is a reduction in development time through the adoption of an 'around the clock' working practice.

This paper introduces a *sequential collaborative software engineering* process involving shift working across time zones and describes an exploratory empirical study of this working pattern involving the implementation of a small-scale software system.

The paper reports on the organisation of the study and on the results obtained through questionnaires, observations and measurements.

1 Introduction

The empirical study described in this paper was carried out as part of a research project investigating the potential for reducing the time needed to carry out software engineering tasks (and hence to deliver software products) through the use of around the clock working. The working style being addressed is one where a task is passed in a sequential manner from one software engineer to another 'across time zones'. We call this sequential collaborative software engineering (SCSE).

A general three-site scenario is illustrated in Figure-1, and shows:

- the *time differences* between sites (which may involve some overlap);
- the *reporting time* when, at the end of a working period or shift, a developer records progress made;
- the *catching up time*, when a developer catches up with the work carried out during the previous shift.

As well as undertaking an empirical study of SCSE, our research has involved the development of a set of equations which models the relationships between estimated development time using SCSE, estimated development time for single-site working, a number of contextual factors and the overheads associated with SCSE [Taweel02].

The contextual factors that are included in the model are:

- the number of sites participating in a collaboration – this is likely to be either two or three sites;
- the time differences between the collaborating sites;
- the number of participating developers;
- the level of concurrency (i.e. the task-specific potential for concurrent working).

The overheads of distribution that are accommodated are:

- extra management, since the added complexity of SCSE over single-site development is likely to require more planning and progress monitoring;
- general task-level communications, which may be needed from time to time for a range of purposes such as reviewing decisions or negotiating the overall approach to be taken;
- reporting time – i.e. the time taken by a developer to report progress at the end of a shift;
- catching up time – i.e. the time taken by a developer to catch up with work completed since completing his/her previous shift;

- distribution effort loss, which is the time lost when a developer at one site fails to complete a task on which a subsequent developer is depending.

The aims of our empirical study were to illustrate the feasibility of employing this type of work pattern and to identify the critical success factors for SCSE. We also expected to obtain some values for the associated overheads (such as reporting time and catching up time) for the particular experimental context.

The remainder of the paper is organised as follows. Section 2 and 3 describe the preparatory activities (such as the selection of subjects, the design of the study and document preparation) and the execution of the selected task. Section 4 and 5 summarise analysis and observations, critical success factors, and final remarks.

2 Preparatory activities

Any empirical study, whether it is a formal experiment, a case study involving a real project, a survey or, as in our case, an informal exploratory study, is a substantial undertaking needing a considerable amount of careful planning [Bausell86, Kitchenham97]. The task is even more challenging when the subjects need to work collaboratively and are to be located in different countries and different time zones.

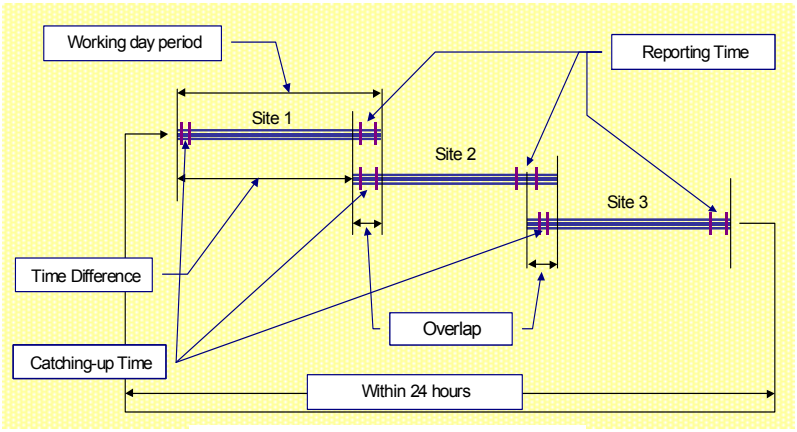


Figure 1: Three Sites

Preparatory activities include the selection of subjects, the overall design of the study and the production of a range of documents needed to support the study and transfer of knowledge between sites.

2.1 Selection of subjects

As for many empirical studies, obtaining subjects is a major problem and this was particularly difficult in our case because of the need to recruit subjects at sites located in different time zones. The sites chosen were Keele (our 'home' site in the UK) and Hebron (in Israel), which have a time difference of 2 hours. We were able to use four subjects (2 in each location) and therefore the SCSE task was executed twice (using 1 subject per site on each occasion). The two sites have similar computing facilities and the subjects have good computing and English language skills.

2.2 Design of the study

The design of the study covers a range of activities including the selection of tasks to be carried out, planning the execution in terms of the procedures to be followed, scheduling and monitoring, deciding what data should be collected and arranging the collection of the data.

2.2.1 The software task

In choosing a software task, a compromise had to be found between the available resources (especially in terms of people and their time) on the one side and the complexity of the task on the other side. Although it may be feasible to distribute other software engineering tasks (such as design and requirements analysis) we felt that the less creative nature of coding (compared especially to designing) would provide greater opportunity for successful collaboration over the necessarily limited time scale of the study. The software engineering task chosen for implementation was a calculator program based on a design that was produced at Keele. The task was selected on the basis that it should be possible to complete the implementation in the time available but that the programming effort would be non-trivial. The requirements specification and the high level design of the chosen software task are shown in Appendix 3

2.2.2 Methods and tools

One essential step was to decide what development methods and tools to use during the study. The approach taken was to choose the method and tools that were familiar to the subjects, in order to avoid the need for training, especially because carrying out training for remote subjects would be very difficult.

These constraints led us to select Java as the programming language and a hybrid of the Yourdon

method [Yourdon79, Coad91] and UML [OMG98, Rumbaugh99] as the modelling language.

2.2.3 The work pattern

Constraints on the availability of subjects led us to schedule the implementation over five shifts each of between two and three hours duration. The time difference between sites meant that the shifts spanned either two or three days. This allowed the subjects in Hebron to fit in their shifts either immediately before or immediately after their normal working hours.

The plan was to give the design document to the subjects a few days in advance and to allow them to ask for clarification from the designer if required.

An initial expected ordering and scheduling of the implementation activities was determined although the subjects were not required to adhere rigidly to either.

2.3 Data collection, knowledge transfer and preparation of documents

Because it was not possible to meet all of the subjects face-to-face, all of the necessary instructions and data collection templates needed to be in electronic form. Several documents were prepared for the different phases of the study:

- *Instructions to subjects* which included an overview of the study and the procedures that should be followed by subjects at each site for each phase;
- *Task description and design* which provided the requirements specification, design and coding convention for the system to be developed;
- *Implementation schedule* proposing an initial allocation of tasks (units of implementation) to subjects over the five shifts;
- *Development dependency chart* describing the dependencies between components of the software system being developed. This document was essential to help subjects understand the various possible orders in which the components could be implemented. This would enable subjects to devise an alternative schedule if they were unable to adhere to the proposed initial schedule;
- *Set-up completion form* to collect information about the documents received by each site and/or problems encountered during the set-up phase;
- *Knowledge transfer template* to be used by subjects during the execution phase to report the status of the task being carried out, enabling them to report problems encountered with received and sent tasks, actions taken by them and progress made. In addition to the code, this template represents the main means of knowledge transfer between sites (see Appendix 5);
- *Timing collection template* to record the catching-up time, reporting time, time spent coding and communication times for each shift;
- *Subject information form* used during the termination phase to collect information about the subjects' experience and personal information;
- *Questionnaire to collect qualitative data* about the subjects experiences and opinions of SCSE.

3 Task execution

The task execution involved three phases: the set-up phase, the execution phase and the termination phase. During the **set-up phase**, documents were distributed to the subjects and any problems addressed. The **execution phase** included the implementation of the software application, collection of quantitative data and progress monitoring. During this phase only source code files and status report of shifts (or implementation units) were transferred between subjects. In the **termination phase**, qualitative data was collected and the software application tested. During the three phases, email was used to transfer documents between subjects and between subjects and the evaluator. Details of a particular shift, which illustrates the activities performed and the information exchanged between sites, are included in Appendix 2.

4 Analysis and Observations

The quantitative data collected throughout the study enabled us to gain some insight into the overheads associated with SCSE in the particular context of the study. Data relating to time spent on the task for each shift, the catching up time, the reporting time and time spent communicating is 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.

It can be seen that on average the total knowledge transfer time (catching up time, reporting time and communications) is 31 minutes. Since the total shift time (on average) was 129 minutes, knowledge transfer made up approximately 24% of the time. Although this proportion is high, it is mainly made up of communication time. When communications facilities are good, as for the Keele site, then the proportion is less than 13%. This overhead works out at less than 9% for a working day of 8 hours under the assumptions that

the communication time remains constant and the catching up time and reporting time increase linearly.

	Overall average across 10shifts (min)	Hebron average across 6 shifts (min)	Keele average across 4 shifts (min)	SD
Development time	98	101	95	-
Catching up time	5	5	3	1
Reporting time	8	11	5	3
Communications time	18	20	6	8

Table 1: Data showing some overheads of distribution

It can be seen that on average the total knowledge transfer time (catching up time, reporting time and communications) is 31 minutes. Since the total shift time (on average) was 129 minutes, knowledge transfer made up approximately 24% of the time. Although this proportion is high, it is mainly made up of communication time. When communications facilities are good, as for the Keele site, then the proportion is less than 13%. This overhead works out at less than 9% for a working day of 8 hours under the assumptions that the communication time remains constant and the catching up time and reporting time increase linearly.

These quite low values for catching up time and reporting time are encouraging especially given the low level of automation provided during the study to support these activities. A greater level of automation is possible and likely if SCSE is used in a commercial setting.

The qualitative data collected was concerned with the subjects' opinions on a number of issues (see 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, subjects felt that their working style was constrained. 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.

In terms of product quality, the overall approach and process followed seems to have had no ill effects. In fact the final systems were fully tested and appeared to have no implementation errors (although one design fault was found).

An additional observation was that 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. Clearly 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'*.

	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)

Table 2: Summary of questionnaire responses

5 Critical success factors

The aims of this work were to illustrate the feasibility of employing SCSE, to identify the critical success factors and to obtain values for the associated overheads (in the context of the study). The values for knowledge transfer time, catching-up time and reporting time are not unduly high and suggest that the overheads associated with SCSE will not prevent it from being a feasible working pattern, In addition, although SCSE was not a popular working pattern, subjects felt that it was efficient and had potential for use in larger projects. In the remainder of this section, we outline the critical success factors.

5.1 Planning

Good planning is crucial for the success of SCSE. Plans need to include both a detailed task schedule and contingencies. Further, it is apparent that timing restrictions imposed by SCSE put software engineers under pressure. This suggests that the distribution pattern with the maximum timing flexibility should be chosen. In addition, although the values of the overheads obtained cannot be generalised it is clear that such overheads are strongly dependent on organisational and management factors. It is important, therefore, to accommodate controlled deviation from planned schedules.

5.2 Documentation

Precise and complete documentation is a critical requirement throughout SCSE as well as being a consequence of the process. A documentation standard must be strictly defined to reduce ambiguities and ensure consistency. Documentation needed includes details of development schedule, requirement specifications, design method and symbols and coding conventions. SCSE also results in the actual process followed being well documented. Any difficulties arising, decisions made, rationales etc. are documented as part of the process and such information could be made available for later analysis if desired.

5.3 Software engineering tasks

Although we expect SCSE to be applicable to different types of software engineering tasks, this study only illustrates its applicability to coding and unit testing. It may be that the more creative tasks that would require a higher degree of collaboration (e.g. design) could be done using SCSE, however, it is likely that such tasks would require substantially better tool support and a great level of synchronous collaboration. The success of SCSE for a software task does not only depend on the characteristics of the software task itself but also on how these characteristics are addressed and considered during the planning and documentation.

5.4 Communications

Not surprisingly, it is essential that the communications mechanisms and tools are easy to use and reliable for SCSE to work successfully.

5.5 Development methods and tools

One of the factors that emerged during this study was the importance of having a compatible set of development methods and tools at participating sites. It was also important that subjects were familiar with their use.

6 Final remarks

In addition to the inherent advantages of SCSE in terms of reduce time scales, we can expect the code produced to be both reliable and maintainable. Three observations lead us to this conclusion:

- the process requires strict adherence to documented coding conventions;
- the code is developed by more than one software engineer so has some of the benefits, such as continuous review, of pair programming [Williams00]);
- every step and phase of development is thoroughly documented (driven by the need for extra planning and supplemented by the information accumulated through the knowledge transfer templates).

Acknowledgements

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

References

Bausell86 Bausell, R. B., *A Practical Guide to Conducting Empirical Research*, Harper & Row Publishers, 1986.

Coad91 Coad, P. & Yourdon, E., *Object Oriented Analysis*, 2nd Ed., Yourdon Press, Englewood Cliffs, N.J, 1991.

Kitchenham97 Kitchenham, B., Linkman, S. & Law, D., *DESMET: a methodlogy for evaluating software engineering methods and tools*, Computing & Control Engineering Journal, Vol. 8, No. 3, pp. 120-126, June 1997.

OMG98 OMG, *Unfiied Modeling Language Specification*, Object Management Group, Framingham, Mass, URL: www.omg.org, 1998.

Rumbaugh99 Rumbaugh, J., Jacobson, I. & Booch, G., *The Unified Modeling Language Reference Manual*, Addison Wesley Longman, Inc., 1999.

Taweel02 Taweel, A., & Brereton, O.P., *Software Development Across Time Zones*, In preparation.

Williams00 Williams, L. and Kessler R. R., Cunningham,W. and Jeffries, R., *Strengthening the Case for Pair-Programming*, IEEE Software, 2000.

Yourdon79 Yourdon, E. & Constantine L., *Structured Design: Fundamentals of a Discipline of Computer Program and Sytems Design*, Yourdon Press, Englewood Cliffs, N.J, 1979.

Appendix 1: Equations for estimating distributed development time

The **gain factors** (GF_{N_s} and GF_{N_e}) in development time (with respect to single site development) are represented by equations 1.0 and 2.0 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).

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

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

Where O_{si} is the overlap between i^{th} site where the i^{th} 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.

The reduction in development time (Reduction Factors) is expressed in the following equations:

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

$$RF_{N_s} = 1 - GF_{N_s} \ , \ 0 < RF_{N_s} \leq 1$$

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

The estimated development time for SCSE is expressed by the following equation:

$$NDT = ST \times RF_{N_s} + PT \times RF_{N_e} + Overheads$$

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 and

PW is the level of potential concurrent working in the given software task.

NDT is the multi-site development time estimate

$$ST = EDT - EDT \times PW$$

where

$$PT = EDT \times PW$$

and

The equations derived are based on the assumption that all utilised sites work within a 24-hour period. This is expressed by the following equation:

$$Total\ working\ period = \sum_{i=1}^{N_s} WD_i - \left(\sum_{i=1}^{N_s - 1} WD_i - TD_i \right)$$

Appendix 2: Details of a shift

The particular shift described is shift two for subject pair B.

At the beginning of the shift the subject received eight files as email attachments. These were:

- the knowledge transfer form (an MSWord document) which was completed by the collaborating subject at the end of shift one;
- the seven source files that had been created during the set up phase and which held all of the coding produced so far. These were made up of four Java source files - Calculator.java, MathFunction.java, Accumulator.java and Operation.java and three VisualCafe project files - Calculator.vep, Calculator.ve2 and Calculator.vpj.

The activities undertaken and the outcomes or results are shown in Table 1 below.

After approximated 70 minutes of the two-hour shift the subject had completed the scheduled activities (see row three in table 1). He had been told that if he finished early he should move on to the activities scheduled for the following shift. However, he did not do this, but instead moved to the 'end of shift' activities.

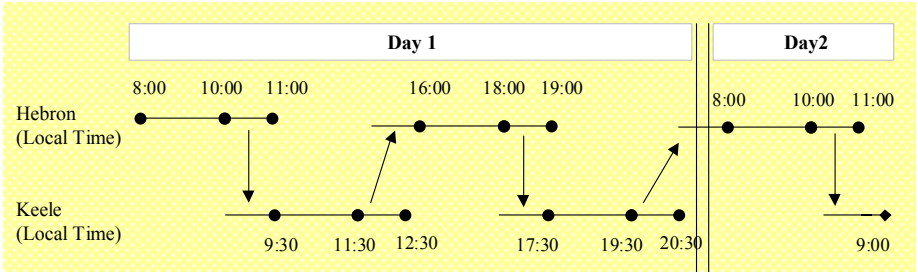


Figure 2: Shift Plan – Subject Pair A

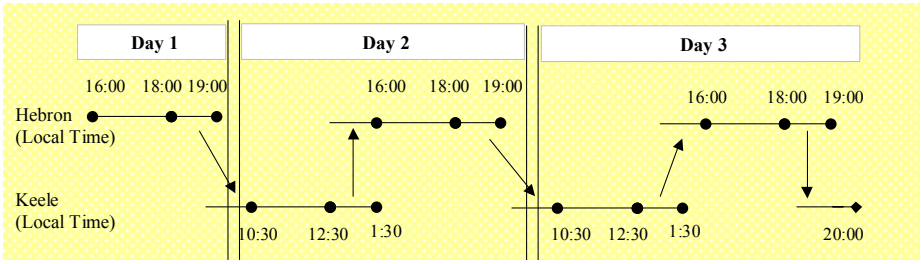


Figure 3: Shift Plan – Subject Pair B

Activity	Outcome/result
1. checked if all necessary documents were received and readable	Yes …. No problems
2. reviewed the knowledge transfer form (see Appendix 5)	One problem noted: *actionPerformed* method could not be fully tested because it calls methods that had not yet been written. To enable testing of all the class, the code written for calling these methods is kept as comments. Subjects at subsequent shifts retest respective code portions of this method when the methods being called are completed (this is part of the unit testing plan outlined in the task description and design document).
3. coded according to the implementation schedule from the point at which the collaborating subject finished	Since the scheduled activities had been completed during shift one this subject worked on methods for the *Accumulator class, Operation class and MathFunctions class*, until all scheduled activities were completed and tested. This shift also involved testing the respective part (that used the methods written in this shift) from the *actionPerformed* method according to the problem highlighted above.
4. performed 'end of shift' activities	Completed a knowledge transfer form, reporting that all files had been received successfully from shift one, all scheduled activities had been performed and no problems had been encountered. All completed tasks were marked with a Y (yes) in the implementation schedule (see Appendix 5). The knowledge transfer form and seven source files were emailed to the collaborating subject (who would work during shift three).

Table 1: Typical Shift Activities

Appendix 3: Requirements specification and class diagram

System Name: **Calculator**

Requirement Specifications:

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

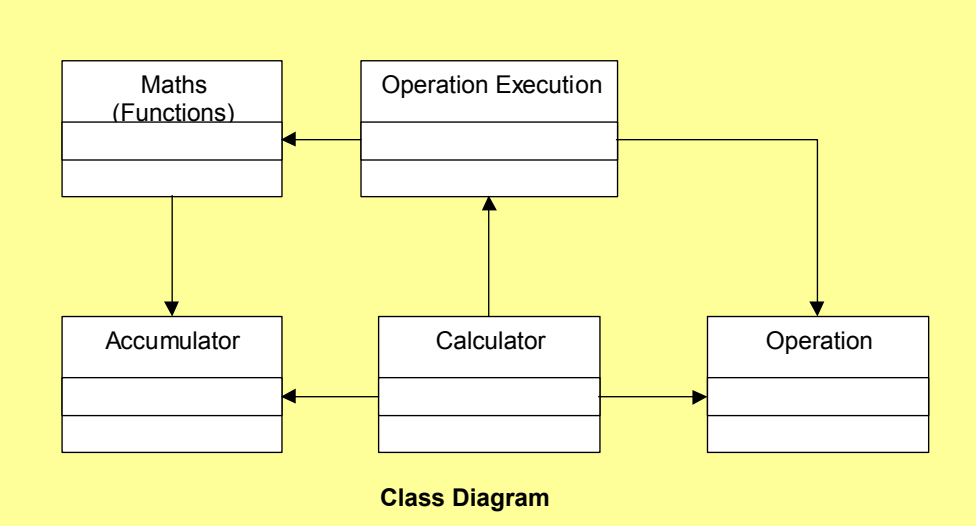
1. An applet based application that runs in a Java-based browser
2. Can be activated using the mouse.
3. Performs the following mathematical functions
   - 3.1. Addition (+)
   - 3.2. Subtraction (-)
   - 3.3. Multiplication (*)
   - 3.4. Division (/)
   - 3.5. Power (x^y)
   - 3.6. Square (x*x)
   -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



Class Diagram

Appendix 4: Implementation Schedule and Dependency Chart

Implementation Schedule

Shift 1

Hebron

Class	Components
Calculator	init() – the interface only
SysAction	All
Display	All

Keele

Shift 2

Keele

Class	Components
Accumulator	All
Operation	All
Math Functions	Add(), Sub(), Div() Power()

Shift 3

Hebron

Class	Components
Math Functions	Mult(), Square(), Percent() SqRoot()
Control Variables	All

Shift 4

Keele

Class	Components
EventHandler	ClearButton() NumbersButton() DecimalPointButton()

Shift 5

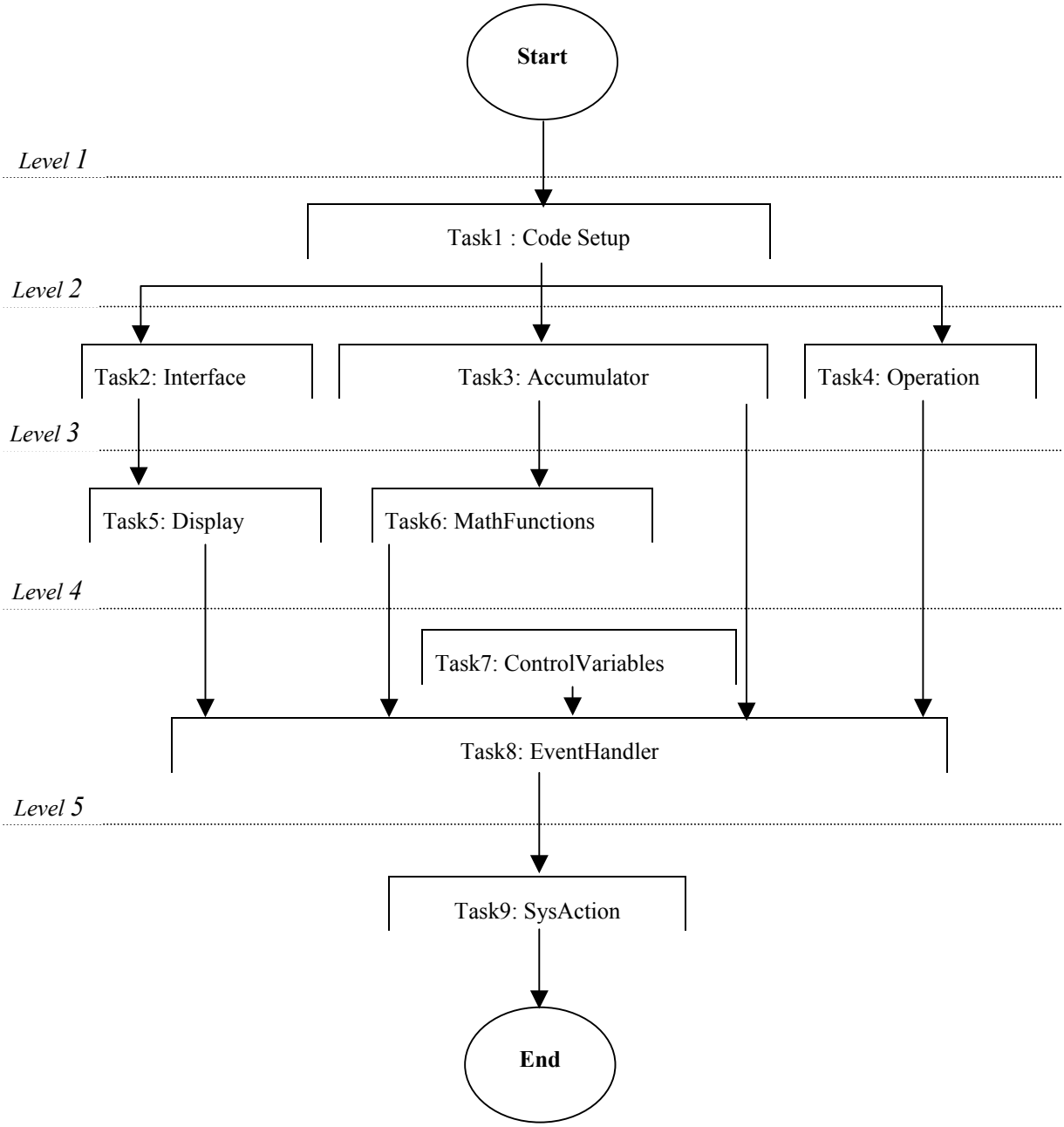
Hebron

Class	Components
EventHandler	ExecuteOperation() OneOperandOperationButton() TwoOperandOperationButton EqualButton()
SysAction	actionPerformed() – Update

Shift 6

Keele

Task	Components
Integration and Testing	All



Development Dependency Chart

Appendix 5: Knowledge Transfer template

Date: xx/xx/xxxx **Shift No** 1

Personal/Site Information

Name: xxxxxx

Site: Hebron **Team Name** **Hebron**

Received Task

Are all received files readable (Yes/No) **yes**

If no, write down names of unreadable files

Any missing components/methods in the received files (Yes/No) **no**

If yes, write down names of missing/expected components/methods

Sent Task

Are ALL scheduled subtasks completed (Yes/No) **yes**

If no, write the uncompleted subtasks/components

Subtask/Method	Why (reason, if any)

Problems encountered (if any) (Yes/No) **yes**

If yes, write down encountered problems

Problem	Description/Suggestion
ActionPerformed method	all methods are kept as comments (because not written yet) but can not test if there is spelling mistakes or any other mistakes.
Display class methods	not called by any other method to test, but I used extra button to call them , they work fine. I deleted the extra code.

Implementation Schedule

Hebron	**Calculator**
	Y init() → Interface
	SysAction
	Y actionPerformed
	Display
	Y UpdateDisplay
	Y ReadDisplay
	Y ClearDisplay
Keele	**Accumulator**
	SetAccumulator
	GetAccumulator
	ClearAccumulator
	Operation
	SetOperation
	GetOperation
	ClearOperation
	MathFunctions
	Add
	Sub
	Div
	Power
Hebron	**Mathfunctions**
	Mult
	Square
	Percent
	SqRoot
	ControlVariables
	SetEnterNewNumber
	SetDecimalPoint
	GetEnterNewNumber
	GetDecimalPoint
Keele	**EventHandler**
	ClearButton
	NumbersButton
	DecimalPointButton
	EventHandler
Hebron	ExecuteOperation
	OneOperandOperation
	TwoOperandOperation
	SysAction (updating)
	actionPerformed
	Integration & testing
	All