An empirical investigation of software project schedule behaviour

Austen William Rainer

A thesis submitted in partial fulfilment of the requirements of
Bournemouth University for the degree of Doctor of Philosophy

July 1999

Bournemouth University
in collaboration with
IBM Hursley Park

Abstract

Two intensive, longitudinal case studies were conducted at IBM Hursley Park. There were several objectives to these case studies: first, to investigate the actual behaviour of the two projects in depth; second, to develop conceptual structures relating the lower-level processes of each project to the higher-level processes; third, to relate the lower-level and higher-level processes to project duration; fourth, to test a conjecture forwarded by Bradac *et al* i.e. that waiting is more prevalent during the end of a project than during the middle of a project.

A large volume of qualitative and quantitative evidence was collected and analysed for each project. This evidence included minutes of status meetings, interviews, project schedules, and information from feedback workshops (which were conducted several months after the completion of the projects).

The analysis generated three models and numerous insights into software project behaviour. The models concerned software project schedule behaviour, capability and an integration of schedule behaviour and capability. The insights concerned characteristics of a project (i.e. the actual progress of phases and milestones, the amount of workload on the project, the degree of capability of the project, tactics of management, and the socio-technical aspects of a project) and characteristics of process areas within a project (i.e. waiting, poor progress and outstanding work). Support for the models and the insights was sought, with some success, from previous research.

Despite the approach taken in this investigation (i.e. the collection of a large volume of evidence and the analyses of a wide variety of factors using a very broad perspective), this investigation has been unable to pinpoint definite causes to explain why a project will or will not complete according to its original plan. One 'hint' of an explanation are the differences between the socio-technical contexts of the two projects and, related to this, the fact that tactics of management may be constrained by a project's socio-technical context. Furthermore, while the concept of a project as a distinct entity seems reasonable, the actual boundaries of a project in an organisation's 'space-time' are ambiguous and very difficult to properly define. Therefore, it may be that those things that make a project difficult to distinguish from its surrounding organisation are interwoven with the socio-technical contexts of a project, and may be precisely those things that explain the progress of that project.

Recommendations, based on the models, the insights and the conclusions, are provided for industry and research.

Overview

Contents

List of figures

List of tables

Acknowledgements

I should like to express my sincere thanks to Prof. Martin Shepperd for all of his support (intellectual and otherwise) throughout the duration of my research. Similarly, sincere thanks to John Allan for his support during the period of full-time research at IBM Hursley Park, and my regular visits since.

I should also like to thank Paul Gibson and Prof. Sa'ad Medhat for laying the foundations between IBM Hursley Park and Bournemouth University which 'bore fruit' with this research project.

Also, I am deeply grateful to the *many* people at IBM Hursley Park (who for reasons of confidentiality must unfortunately remain anonymous) for allowing their projects to be studied (or at least be candidates for study!) and for willingly making time to openly discuss their opinions of their projects and of my interpretations of their projects. This research really would not have been possible without them.

Thanks to Dr. Frank Milsom and Colin Kirsopp for their advice on my research, particularly to Colin for his contributions to the statistical analysis discussed in Appendix A0.

Finally, thanks to Glyn, Si, Tom, Emma, Pete and Min for their friendship over the last four years.

Chapter 1 Introduction

1.1 Statement of the problem

A software product that is delivered to the market earlier than its competitors typically enjoys several advantages over those competitors: the product is seldom obsolete any sooner, has an increased share of the market and a higher profit margin ([84, 114])[1].

At the same time, there is a substantial amount of research to show that software development projects are frequently completed later than planned. Some of this research surveys a broad range of projects (see, for example, [10, 21, 35, 57, 123]) whilst other research concentrates on particular projects (e.g. [63, 80, 95, 134]), some of these projects being of high public interest, such as the London Ambulance Service's Computer Aided Dispatch System (e.g. [8]).

The frequency of poorly performing software projects suggests that project managers have great difficulty both planning and executing their projects. This may be for a number of reasons, such as:

- Project managers may lack a comprehensive and widely-applicable understanding of the behaviour of software projects. Their lack of understanding is increasingly likely as products become increasingly complex and as they address new, increasingly complex requirements. Obvious examples are projects that are involved with rapidly developing technologies, such as the World Wide Web.
- Events beyond the control of the project prevent these projects completing according to plan.
- The goals of the project, particularly the product's requirements, cannot be stabilised, with the result that the project either completes later than planned or may even be abandoned.
- The demand for new products, and the competition between products within a market, creates 'pressure' for increasingly shorter project durations. This may cause project managers to take increasing risks with their projects, with the consequence that projects are increasingly likely to complete later than planned.

Two fundamental goals of research are first to explain, and then to communicate that explanation, so that practitioners may make better informed decisions and take better informed actions. Software engineering research has yet to provide a comprehensive

[1] Not all products first to the market are, however, the eventual 'winners' (e.g. [16]).

explanation of the behaviour of software development projects and, more particularly, the factors affecting project duration. For example, Carmel ([20]) writes:

> "It should be noted that nowhere does the software engineering literature make any causal claims regarding cycle time. Instead, the variables are normative and prescribed for 'successful development'." ([20], p. 112)

Taking a broader perspective, Olsen ([84]) complements the opinion of Carmel. Olsen writes:

> "Not only does engineering literature rarely address time-to-market as the central goal, but management often embrace the short-sighted view that the most important goal is to control software labor and capital budgets, without considering the effect on time-to-market. This is often because information engineers cannot persuasively show how cost factors affect time-to-market; they typically have few tools and case histories to back up their recommendations - whereas budget costs are all too clear." ([84], p. 30)

Abdel-Hamid and Madnick ([2]) argue that the software engineering research community still lacks a fundamental understanding of software development processes. In a related field to software engineering research, that of information systems (IS) research, Remenyi and Williams ([101]) argue that there is no established theory, and Jarvenpaa ([55]) argues that the lack of theory development, rather than appropriate research methodologies, is the real problem for IS research.

On the subject of the *development* of theory, Eisenhardt ([39]) draws upon Glaser and Strauss ([45]) to argue that:

> "... it is the intimate connection with empirical reality that permits the development of a testable, relevant, and valid theory." ([39], p. 532)

Thus, a testable, relevant and valid explanation of software project schedule behaviour is one founded on a close, solid connection with the *actual* processes of software development and the processes of managing that development. This necessarily requires a focus on particular projects, so that the subtleties, nuances and complexities of actual process may be best understood. (A focus on particular projects raises the problem of applying the findings drawn from these projects to a broader set of projects. This problem is briefly considered in section 1.4, and then considered in more depth in chapters three and nine.) Given that Carmel and Olsen are correct in their assessment of a lack of

research explaining software project cycle-time, then one fundamental reason would appear to be a lack of research that seeks to *generate* these explanations i.e. a lack of studies that intimately connect with empirical reality. There are a number of bodies of research, within the software engineering community, that are potentially relevant to the development of theory. These bodies of research and the issue of generating theory are discussed in more depth in chapter two.

Overall, there is a need for explanations of software project behaviour and, more specifically, software project schedule behaviour, and while empirical evidence and conceptual structures do exist there is no established theory.

1.2 A definition of 'software project schedule behaviour'

The term 'software project schedule behaviour' is meant to convey the following:

- The duration of a project from its initiation to completion.
- The duration of a project from its initiation to the delivery of its product (the product may be delivered before the project is completed).
- How those durations are structured and associated with work i.e. the project schedule. These structures and associations consist primarily of intervals of time (i.e. phases) and instantaneous events (i.e. milestones).
- How those structures and associations change during the project i.e. the dynamics of the schedule.
- How those dynamic structures and associations are interwoven with the wider behaviour of the project.

1.3 Aims of the inquiry

Given the need for, and lack of, explanation this inquiry has the following aims:

1. To consider the degree to which existing empirical studies within the software engineering research community identify, describe or explain relationships between the actual processes of software development and the schedule behaviour of software projects.
2. To identify gaps within the existing research that prevent, or limit, the development of a theory.
3. To identify the opportunities for a contribution in this area of research, and to select one or more of these opportunities as specific objectives for the empirical component of this research.

4. To conduct empirical inquiry, so as to contribute to the body of research on software engineering in general and software project schedule behaviour in particular.

1.4 Scope of the inquiry

In order to make this inquiry feasible, the inquiry is bounded in a two ways. First, as already suggested, this inquiry concentrates on extant knowledge within software engineering research. This is recognised as a potential limitation to this inquiry, and as a result *some* attention is directed outside of software engineering research. (This attention is mainly directed, however, at methodological issues.)

Second, this inquiry places particular value on Eisenhardt's requirement for an intimate connection with empirical reality. This has two implications. First, some potentially valuable research, such as experts' anecdotal accounts of software projects (e.g. [51] and [136]), are excluded because they do not *communicate* systematic and detailed evidence on actual processes. (In this context, anecdotal accounts are distinguished from the narrative accounts provided by ethnographic studies.) Second, and as noted previously, an emphasis on particular processes introduces the problem of applying findings to other projects. This investigation seeks to overcome this problem in two ways. First, having conducted the empirical component of this inquiry, attention is re-directed at previous research to determine whether other studies have independently drawn similar insights. Second, part of the empirical component of this inquiry seeks to test a conjecture made by Bradac *et al.* ([18]) and thus strengthen the applicability of that conjecture to a wider set of projects.

1.5 Structure of the thesis

The remainder of this thesis is organised as follows. Chapters two and three lay the theoretical and methodological foundations for the subsequent empirical inquiry. Chapter two concentrates on the contribution of studies of actual time usage in software projects for explaining software project schedule behaviour. This discussion includes a particular consideration of the work of Bradac *et al.* ([18]) as part of his work is tested in the subsequent empirical inquiry. Chapter two also identifies specific opportunities for further research and appropriate objectives for the empirical investigation conducted as part of this research.

Chapter three discusses a collection of methodological issues. The chapter identifies the appropriate research strategy (the case study research strategy) to achieve the objectives identified in chapter two, discusses the selection of cases, the volume of evidence

collected, the types of analyses conducted on that evidence, and various operational details of the case studies.

Chapter four presents and discusses two models that were iteratively developed from the evidence. The first model is a simple model of software project schedule behaviour. The second model is a model of capability. The chapter shows how these two models can be integrated into a third model, and how they can also be related to the studies of actual time usage discussed in chapter two. Chapter four also discusses a number of caveats and problems with the models, as well as potential alternatives to the models.

Chapters five through nine present and discuss the behaviour of the two projects that were studied. Chapter five presents comprehensive analyses based around the model of software project schedule behaviour. The scope of the inquiry is broad, considering the socio-technical contexts of the two projects, the actual progress of the two projects, and the management tactics used by the two projects. Chapters six through eight each examine one characteristic of process areas of a project, using the model of capability. Chapter six examines reports of waiting. Chapter seven examines reports of the progress of work (and particularly the poor progress of work). Chapter eight examines reports of outstanding work. (Chapter five provides a context within which the analysis of these three characteristics can be better understood.) Chapters six through eight include a test of Bradac *et al.*'s ([18]) conjecture that waiting is more prevalent during the end of a project than during the middle of the project.

Chapter nine then brings together the various 'threads' of chapters two and four through eight. The chapter relates the insights drawn from chapters five through eight with the models presented in chapter four and the review of actual time usage presented in chapter two. Chapter nine also presents a second review of previous research, this review focusing on the specific insights gained from the empirical inquiry. Finally, chapter nine speculates on the wider applicability of the insights.

Chapter ten then summarises the investigation, considering the components of the empirical analyses, the main conclusions, some recommendations, threats to the validity of the conclusions and opportunities for further research. Chapter ten also reviews the aims of the inquiry and the degree to which they have been satisfied.

The appendices provide detailed empirical evidence on the two projects. For reasons of confidentiality, transcripts of the interviews and the content of the minutes of status meetings are not included.

Chapter 2 Actual time usage in software projects

2.1 Introduction

As explained in chapter one, particular value is placed on Eisenhardt's ([39]) argument that the generation of explanation requires an intimate connection with empirical reality. The argument was also made that there appears to be a lack of research, within the software engineering community, that seeks to generate explanation.

This chapter first briefly reviews a number of bodies of research, within the software engineering community, to support the argument that there is a lack of theory-generating research. The chapter then concentrates on reviewing studies of actual time usage in software development projects. In principle, studies of time usage are considered to be an excellent method for an intimate connection with empirical reality because they (potentially) explore both 'visible' and 'invisible' work ([82]). Also, studies of actual time usage provide the most direct connection with intervals of time and instantaneous events in a software project.

2.2 A brief review of five areas of software engineering research

Besides research on time usage, five bodies of research have been identified as potentially relevant to the development of explanations of software project behaviour and software project schedule behaviour. These are;

- Surveys of practitioners' opinions of the software process.
- The development and validation of system dynamic models of software development projects.
- The development and validation of prediction systems of characteristics of software projects e.g. effort, cost, quality and duration.
- The development and validation of software process models.
- Investigations of actual process.

Surveys (e.g. [9, 10, 21, 35, 38, 47, 74]) investigate tendencies but are not ideally suited to explaining those tendencies.

System dynamics is a promising approach to both explaining and predicting the behaviour of software development projects, but there appears to be little substantial empirical

work, within the field of software development, available to-date. The main empirical research is provided by Abdel-Hamid (e.g. [2, 3, 108]) with more recent contributions by Lehman *et al.* (e.g. [68-70]) and Tvedt ([126, 127]). Other recent contributions appear to be mainly theoretical in content (e.g. [22, 27, 28, 103-105, 130]).

Prediction systems (e.g. [50, 60, 61, 65, 73, 106, 110, 111, 129, 135]) develop relationships between measured-attributes and predicted-attributes, but these relationships are not prescribed as causal relationships, and are not assumed to provide explanation.

The development and validation of software process models is founded on the logic that improving the processes of development will improve the outcomes of that development e.g. reduced project cost, effort and duration, and increased product quality, functionality and performance. In considering this area of research, Rodden *et al.* ([102]) first characterise this research as being typically concerned with developing (or validating) abstract descriptions that are to be *instantiated* for a particular (organisational) setting before being *enacted* to manage the use of tools within an environment. Rodden *et al.* then argue that these abstract descriptions are often *too* abstract in that they no longer (or, perhaps, at no time did) represent the actual *nature* of software development. (It may be that software process research is first concerned with developing appropriate technologies before using those technologies to inquire on the actual process.)

Although the tendency in software process research is toward the development of abstract models of process, there are a number of studies of actual process. Many of these studies, however, do not relate their findings explicitly to software project schedule behaviour. Also, studies of actual process tend to investigate the lower-level processes, such as individuals, teams and activities (e.g. [18, 25, 26, 32, 48, 49, 71, 85, 91, 92, 95, 112, 113, 115, 128, 132, 138]), rather than the higher-level processes, such as 'functional areas' of the project, the project itself, and the organisation 'surrounding' the project. (It may be that the focus on lower-level processes reflects difficulties with investigating actual software development processes in-the-large.) Curtis *et al.*'s ([31]) seminal study of large software systems is perhaps the only study that seeks to empirically investigate the interactions between the various process levels of a project. Their study does not, however, explicitly relate the effect of these interactions on software project schedule behaviour. Watson ([134]) reports on the use of COCOMO (e.g. [13, 14] see also, more recently, [12]) as a tool for validating estimates made though other methods. *Implicit* within his study is an examination of processes at a high-level i.e. the major phases of the project. Watson's study is considered in depth in chapter nine.

The position taken in this thesis is that while all of these bodies of research are valuable for the long-term development and validation of theory, they are currently not ideally suited to generating the initial material for an explanation. (This is comparable with Jarvenpaa's [55] claim that the lack of theory development is the real problem for IS research.) As already noted, the remainder of this chapter concentrates on reviewing studies of actual time usage. Chapter nine complements the review in this chapter by reviewing some of the research identified above in light of the insights gained from the empirical component of this inquiry.

2.3 Studies of actual time usage in software projects

There appears to be few studies that specifically investigate the characteristics and effects of time usage in software development projects. (i.e. [5, 18, 34, 92]). Of these, Bradac, Perry and Votta's study ([18]; an earlier version was published as [17]) appears to be the first study conducted in this area.

Bradac, Perry and Votta's 'prototype experiment'

Bradac *et al.* ([17, 18]) conduct a study to investigate what people actually do when they add features (features are sets of market requirements) to a large software system. Their study is a prototype study, conducted as preparation for a more substantial subsequent study ([92]) that consists of a time-diary study and a direct-observation study. The findings of the prototype study and the subsequent study establish some assumptions that inform two further studies ([5, 34]).

Table 2.3.1 Bradac *et al.*'s frequency of states

State	% time
Working the process	19.6
Documentation	8.2
Reworking the process	7.0
Reworking the documentation	4.2
Waiting on the laboratory	2.7
Waiting on an expert	3.1
Waiting on a review	9.2
Waiting on hardware	1.0
Waiting on software	1.9
Waiting on documentation	2.4
Waiting on other	40.7
Total	100.0

Bradac *et al.* initially characterise the software process in terms of fifteen tasks and eleven states, with the states referring either to some type of progress or to some type of waiting. One of their "basic set of analyses" ([18], p. 781) is to investigate the frequency

of states, and the results of this analysis are reproduced here in Table 2.3.1. As the table indicates, about 40% of the time spent in the process is spent being productive (the first four states in the table) and 60% of the time *appears* to be spent waiting. The apparent frequency of waiting is discussed below. Using the information presented in the table, Bradac *et al.* form a conjecture that one important way of reducing the development interval is to significantly reduce the number of days in blocking states. They add, however, that the 40:60 ratio is dependent on the concurrency of processes. If the global process (presumably, Bradac *et al.* mean the project-level process; they do not provide an explicit definition) also experiences a 40:60 ratio (which would be affected by the concurrency of processes) then significantly reducing the amount of time spent in blocking states would significantly reduce project duration.

Table 2.3.1 indicates that the 'Waiting on other' state clearly dominates the frequency of states, and it is the dominance of this state that raises the issue of whether waiting actually occurs for 60% of the process. Bradac *et al.* recognise that the dominance of the 'Waiting on other' state reveals a weakness in their characterisation, and this leads them to revise their characterisation so that their states refer either to some type of working, to some type of waiting, or to some type of not working. The not working categories are:

- Not working, training
- Not working, reassigned
- Not working, vacation
- Not working, weekend
- Not working, other

Having revised their characterisation, it is unfortunate that Bradac *et al.* do not then present details of the frequency of their revised set of states, because their reworked classification provides a different interpretation of their findings. This is discussed below. (It is understandable that Bradac *et al.* do not report on their revised classification when one considers that the revised characterisation is a *result* of their prototype study, intended to be used in subsequent studies.) In Perry *et al.*'s study ([92]), the subsequent study to Bradac *et al.*'s study, it is clear that the five most prominent states of the revised classification are, in descending order:

1. Not working, reassigned
2. Not working, weekend
3. Working the process
4. Not working, other

5. Reworking the process.

It is clear from Perry *et al.*'s evidence that the not working categories, and particularly the 'Not working, reassigned' category, occur more frequently than the waiting categories. (It is difficult to establish accurate breakdowns from the information provided by Perry *et al.*) This evidence suggests an alternative interpretation of Bradac *et al.*'s evidence, *viz.* that the not working categories potentially have more affect on the process than blocked work. The effect of vacations and weekends ought to be planned for, so this leaves the effect of the 'Not working, reassigned' category as particularly interesting. It is interesting because one of the techniques that management use to manage projects is to reassign work. Waterson *et al.* ([133]) found, for example, that workload fluctuated and that teams would be temporarily restructured (with staff being drafted in from other teams in the project if the workload became too demanding) to ensure that project milestones and deadlines were met. In principle, the reassignment of work may then cause problems elsewhere in a project because resource is drawn away from those parts of the project.

The frequency of time spent in the 'Not working, reassigned' category in Perry *et al.*'s paper (and potentially Bradac *et al.*'s paper) suggests that other parts of the project, or *other projects*, are experiencing problems. Once again it is unfortunate, although entirely understandable, that evidence on the project as a whole and the wider organisation is not available from Bradac *et al.*'s and Perry *et al.*'s papers. Despite this lack of evidence, Perry *et al.* ([92]) recognise that developers may be reassigned to higher priority projects, and for Perry *et al.* the reassignment of developers to other work reflects the fact that large-scale software development projects are extremely dynamic.

The reassignment of developers to higher priority projects supports the argument that a project is affected by factors external to the project (see, for example, [11]). The 'twist' here is that the effect consists of drawing away resources to external projects rather than imposing requirements, or constraining the project through technical and strategic dependencies with other projects.

In addition to their observation on the breakdown of states, Bradac *et al.* observe that:

> "... blocking tends to be more prevalent at the beginning and at the end of the process." ([18], p. 783)

This observation leads Bradac *et al.* to conclude that one should attack the blocking factors in the requirements, high-level design and high level test phases of the process. This also suggests that the requirements, high-level design and high-level test phases have

the most influence on software project schedule behaviour. Bradac *et al.* do not define what parts of the project constitute the beginning and the end of a project (see chapter three for more information). They do express an interest as to whether this conjecture is valid for a wide variety of projects.

Subsequent research to Bradac *et al.*'s study

Bradac *et al.* advise caution on how their findings should be treated. They write:

> "We reiterate our caveat about this data: Though they are real data, they are reconstructed data of only one instance of the process, with some blurring of the accuracy because of retrospection. We feel, however, that there are some intriguing *conjectures* about our feature development processes that we hope to *validate with subsequent experiments*." ([18], p. 783; emphasis added)

Indications that these subsequent experiments occur are the publications by Perry *et al.* (e.g. [92]) that report on the conduct of the time-diary study and the direct-observation study, a publication by Ballman and Votta ([5]) that reports on *simulations* (rather than investigations of actual behaviour) of meeting congestion, and two publications by Dandekar *et al.* ([33, 34]) that report on a process simplification exercise.

Perry *et al.*'s study ([92]; but see also [91, 93]) appears to confirm the 40:60 ratio, for a designer, between being productive and waiting. Their evidence may also be used to support the alternative interpretation raised above *viz.* that the 'Not working, reassigned' category potentially has more of an effect on the process than waiting.

Ballman and Votta ([5]) develop a model which relates the average waiting time for a meeting to the number of meetings in developers' calendars. As the fraction of the 'population' of developers in the organisation required to attend the meeting increases, and as the meeting generation rate increases, so the time between when the meeting was arranged and when it can occur increases in a non-linear relationship. From this Ballman and Votta argue that:

> "A significant portion of an individual feature interval seems to consist of time lost while developers wait for meetings" ([5], p. 123)

It must be emphasised, however, that Ballman and Votta simulate the effect of meeting congestion on *the time until a meeting actually occurs*, and that they then use the results of their simulations to *imply* that the time until a meeting occurs affects feature interval.

Overall, Ballman and Votta's investigation provides insights into the potential effects of a lower process (i.e. scheduling meetings) on a higher process (i.e. the behaviour of a feature's development process) and so complements rather than validates the work of Bradac *et al.* and Perry *et al.*

Dandekar *et al.* ([33]) also draw upon the work of Bradac *et al.* ([17]) and Perry *et al.* ([92]) when they write:

> "One of the primary problems in large-scale software development is the time spent waiting for resources, responses, meetings, etc. One may be able to fill in the intervening time productively, but for a particular sequence of activities there may be a significant difference between the actual time spent and the time that elapses before completion... " ([33], p. 3)

A closer examination of Bradac *et al.*'s paper indicates that Bradac *et al.* provide evidence consisting of a time-line lasting 75 days for one developer, in which there were three instances of waiting on reviews (lasting between four and seven days) and two instances of waiting on experts (one lasting two days and the other lasting three days). While this empirical evidence is intriguing, it does not seem to constitute a body of evidence that is conclusive. Strictly speaking, Bradac *et al.* do not provide sufficient evidence from which Dandekar *et al.* can claim that waiting is a primary problem in large-scale software development. (It may be that because Bradac, Perry and Dandekar are all researching within the same organisation, Dandekar *et al.* have access to evidence unpublished by Bradac *et al.*, perhaps unpublished for confidentiality reasons.) Rather than validating Bradac *et al.*'s claims, Dandekar *et al.* assume those claims to be valid and use them to direct their own research.

All four of the studies ([5, 18, 34, 92] reviewed here were conducted at Lucent Technologies and, so far as this author is aware, there are no independent studies that seek to corroborate the findings of these studies. Thus, there is no independent support for their conclusions. With this in mind, it is particularly important to emphasise Bradac *et al.*'s caution with regards the quality of their evidence and how it should be treated.

It is also important to re-iterate the distinction between the time-usage of an individual designer and the time-usage of the project, and the caveat that the project's time-usage must exhibit the same 40:60 ratio as the designer's time-usage for one (and possibly more) of Bradac *et al.*'s conjectures to apply. Perry and his colleagues do not appear to have subsequently investigated this caveat for the phenomenon of waiting, although Ballman and Votta's study provides some support for the caveat.

Van Genuchten's investigation of activities

Two years prior to the publication of Bradac *et al.*'s investigation, van Genuchten ([128]) published the findings of a related study. Whereas Bradac *et al.* focus principally on intervals of time, and how those intervals of time are used (i.e. the states of working, waiting or not working), van Genuchten focuses on activities and how much time they use. Because of their contrasting perspectives, Bradac *et al.* are able to investigate 'invisible' tasks (*cf.* [82]) and visible tasks, whereas van Genuchten investigates only visible tasks. Van Genuchten's study complements Bradac *et al.*'s study because both studies investigate time usage, and the effect of waiting, but in different ways.

Van Genuchten ([128]) investigates reasons for why activities start later or earlier than planned, why these activities last longer or shorter than planned, and why the effort expended on activities is more or less than planned. Van Genuchten collects evidence on 160 activities across six representative projects in one software development department. The average duration of an activity is four weeks and the average effort is approximately 100 person-hours. Van Genuchten discourages unjustified generalisations from his findings, providing evidence from another department to demonstrate that the distribution of reasons varies strongly for different software development departments.

Table 2.3.2 presents a simplified version of van Genuchten's classification of reasons, together with an interpretation of these reasons from the perspective of waiting. All six categories used by van Genuchten are shown in the table, but only those relevant to this discussion are elaborated. One should be cautious about interpreting van Genuchten's reasons as types of waiting when his study was not conducted with that perspective.

Table 2.3.2 Van Genuchten's classification of reasons

Code	Capacity-related reasons Reason	From a blocked work perspective
11	Capacity not available because of overrun in previous activity	Waiting on capacity to become available
12	Capacity not available because of overrun in other activity	Waiting on capacity to become available
13	Capacity not available because of unplanned maintenance	Waiting on capacity to become available
14	Capacity not available because of unplanned demonstration	Waiting on capacity to become available
15	Capacity not available because of overrun in other unplanned activities	Waiting on capacity to become available
16	Capacity not available because of overrun in other causes	Waiting on capacity to become available
19	Other	Waiting on capacity to become available
	Personnel-related reasons *not elaborated*	
Code	**Input-related reasons** **Reason**	**From a waiting perspective**
31	Requirements late	Waiting for requirements
32	Requirements of insufficient quality	Waiting for requirements of sufficient quality
33	(specifications of) delivered software late	Waiting for (specifications of) delivered software
34	(specifications of) delivered software of insufficient quality	Waiting for (specifications) of delivered software of sufficient quality
35	(specifications of) hardware late	Waiting for (specifications of) hardware
36	(specifications of) delivered hardware of insufficient quality	Waiting for (specifications of) delivered hardware of sufficient quality
39	other	
	Product-related reasons *not elaborated*	
	Organization-related reasons *not elaborated*	
Code	**Tools-related reasons** **Reason**	**From a waiting perspective**
61	development tools too late or inadequately available	Waiting on development tools
62	test tools too late or inadequately available	Waiting on test tools
	Other *not elaborated*	

It is clear from van Genuchten's study that of those activities that start late, approximately 80% of them start late for capacity-related reasons. Van Genuchten explains that this was because many activities start late because of a delay in completing a previous activity (i.e. reason 11). From a perspective of waiting, many activities start late because they are waiting on a previous activity to complete, or waiting on resource to become available from a previous activity. Van Genuchten also seems to suggest that unplanned work, particularly unplanned maintenance work (i.e. reason 13), is an important reason for activities starting later than planned. Clearly such unplanned activities mean that resource becomes unavailable to conduct planned activities because the resource is addressing unplanned activities.

For the remaining activities that start late, they start late either because of input-related reasons (approximately 15%) or because of tools-related reasons (approximately 5%). As Table 2.3.2 indicates, all of the input-related and tools-related reasons can be interpreted as types of waiting. The implication is that for those activities that start late, they *all* start late because they are waiting on something.

With regards to the reasons for differences between the planned and the actual durations of the activities, input-related reasons account for differences in 20% of the cases, and capacity-related reasons account for differences in almost 40% of the cases. In total, approximately 60% of the differences between planned and actual durations are due to instances of waiting. Van Genuchten does not identify whether these differences were 'positive' (i.e. the actual duration was greater than planned) or 'negative' (actual duration was less than planned) but it seems unlikely that durations are shorter than planned because of waiting.

With regards to the planned and actual effort, Van Genuchten finds a prevalence for when in the project actual effort increased over the planned effort. He writes:

> "... the relative differences between planned and actual efforts increased for the subsequent phases of the projects..." ([128], p. 587)

and for the planned and actual starts and durations of activities, Van Genuchten writes:

> "... the delays [to the start of activities] and overruns [in the duration of, and the effort for, activities] increased toward the end of the project." ([128], p. 587)

This observation is consistent with Bradac *et al.*'s observation that waiting is more prevalent at the beginning and at the end of a project.

Summary of research into time usage

Table 2.3.3 Summary of studies that investigate time usage

Investigation	Unit of analysis	Cases	Method
Bradac *et al.* [17, 18]	time segment	one developer	time-diary
Perry *et el.* [91-93]	time segment	13 developers across four departments	time-diary & direct-observation
Ballman & Votta [5]	meetings	three artificial 'projects'	simulation
Dandekar *et al.* [33, 34]	activities	the inspection process	value added analysis, time usage, alternatives analysis
van Genuchten [128]	activities	six projects in one department	activity analysis

Overall, it appears that only a small number of studies have investigated actual time usage in software development projects. These studies are summarised in Table 2.3.3. The studies concentrate on describing the use of time at the lower levels of a project. None of the studies explain the effects of time usage on project duration, although some of the studies speculate what the effects might be. The lack of explanation is significant because these studies, and this thesis, assumes that explaining actual time usage is an important foundation upon which to explain project duration.

All of the studies employ a case-based research strategy, focusing on one or a few cases. This raises concerns as to the applicability of the findings to other projects; a concern recognised by Bradac *et al.* when they advice caution on how their findings should be treated. (Bradac *et al.* also express an interest as to whether their conjectures would apply to a wider set of projects.) All of the studies employ *a priori* classifications or models to analyse their evidence.

With regards to the content of these studies, the studies distinguish between working, not working and waiting states. Bradac *et al.* find that waiting accounts for 60% of the time spent in the process (at a low level). An alternative interpretation is that not working states, specifically the state of being reassigned to another project, may account for much of this time. The frequency of time spent assigned to other projects then suggests that

external factors are an important influence on a project. Bradac *et al.* also investigate when waiting is most prevalent, and suggest that reducing waiting during the requirements, high-level design and high-level test phases are likely to be the most effective areas for reducing project duration. Bradac *et al.* also outline the requirement for the project-level processes to exhibit the same 40:60 ratio as the low-level processes for blocked work to affect project duration.

With one exception, all of the studies were conducted at Lucent Technologies, and they concentrate on the possible impacts of blocked work. Only van Genuchten's study has been conducted elsewhere and his study does not replicate, but rather complements, the Lucent Technologies' studies. Furthermore, the Lucent Technologies' studies subsequent to Bradac *et al.*'s study do not validate the observations of Bradac *et al.* but rather assume them to be valid, and conduct further research based on those assumptions. There is, therefore, a clear need for independent replication of the above studies, or testing of the conjectures from those studies.

2.4 Opportunities and objectives for this research

Opportunities

The summary presented at the conclusion of the preceding section suggests a number of opportunities for subsequent investigations. The first opportunity is to partially or completely replicate one or more of the studies reviewed. The most valuable study to replicate would be Bradac *et al.*'s study as this is the first study in the area, has not been replicated, and has established assumptions upon which subsequent studies are founded. A successful replication would strengthen both the claims of Bradac *et al.*'s study and the claims of the subsequent studies that have built on Bradac *et al.*

A second opportunity is the investigation of higher-level processes. All of the studies of actual time usage have looked at the lower-level processes and there are no studies that have looked at the higher-level processes within the context of schedule behaviour. With studies of higher-level processes, the concept of 'time usage' becomes more abstract as the inquiry is no longer concerned with how individual's actually use their time, but rather how teams, process areas and the project actually uses time.

A third opportunity is the investigation of the effects of the lower-level processes on schedule behaviour. This would require a research design that investigated both the lower- and higher-levels of the process. There appear to be no studies that have looked at lower-

and higher-levels of the process within the context of software project schedule behaviour.

A fourth opportunity is the development of a theory of software project schedule behaviour. As argued in chapter one, there is no established theory of software project schedule behaviour, so the provision of a theory is desireable. The lack of studies of the higher-level processes, and of the relationships between lower-level and higher-level processes indicates that the development of a theory is premature.

Finally, given that the development of a theory of schedule behaviour is premature, there is the opportunity to develop 'conceptual scaffolding' ([141]). As with physical scaffolding, the purpose of conceptual scaffolding is to support 'construction', in this case the construction of a theory. Conceptual scaffolding should not be confused with the conceptual structure itself, and upon completion (or nearing completion) of the conceptual structure, the scaffolding may be 'thrown away'.

Objectives

Given the opportunities available for investigations in this area, the value placed on Eisenhardt's requirement for an intimate connection with empirical reality, and the desire to contribute to the systematic accumulation of evidence, the objectives for the empirical component of this investigation are:

1. To replicate parts of Bradac *et al.*'s study. This will consist of a replication of the types of waiting, and a test of Bradac *et al.*'s conjecture that waiting is more prevalent during the end of the project than during the middle of the project.
2. To investigate actual time usage at higher-levels of the project, specifically at the level of the whole project, and also at the level of process areas within the project.
3. To investigate the relationships between the lower-level and higher-level processes, and their relationships to schedule behaviour.

The second two objectives will be addressed in two complementary ways. First, narrative descriptions and explanations will be provided for both the characteristics of the project and the process areas within the project. Second, more formal descriptions and explanations, in the form of conceptual models, will be provided.

Chapter 3 Methodology

3.1 Introduction

This chapter discusses four methodological issues:

1. An appropriate research strategy for developing descriptions and explanations of software projects and their schedules.
2. Heuristics for the design and conduct of case studies.
3. Some technical details relating to the selection of cases, the identification and exploitation of appropriate sources of evidence, and the development of methods for analysing the evidence in order to describe and explain behaviour.
4. Some operational details concerning the organisation of evidence relating to process areas, and the replication of a part of Bradac *et al.*'s study.

Points 3 and 4 concern detailed information that might normally be explained together with the actual analysis of the evidence (i.e. in chapters four through eight). This information, however, is common to all the subsequent chapters (and underpins much of the empirical analyses) and consequently it is more efficiently explained once here.

3.2 An appropriate research strategy

Because of the desire to develop descriptions and explanations of actual behaviour, particular value is placed on Eisenhardt's ([39]) argument regarding the need for an intimate connection with empirical reality. From this argument follows a requirement for the systematic study of actual processes and, by implication, the study of particular processes.

Of the three broad strategies for collecting and analysing empirical evidence (i.e. experimental study, survey study and case study) it is widely recognised that the case study research strategy is most appropriate for investigating particular real-world settings.

Benbasat *et al.* ([7]) draw upon a number of previous researchers' arguments and definitions (i.e. [6, 15, 58, 119, 139]) to provide a useful summary of the characteristics of a case study. These characteristics are presented here in Table 3.2.1. Simplifying the points presented in the figure, a case study is an intensive investigation of one or more entities within their natural setting.

Table 3.2.1 Benbasat *et al.*'s characteristics of a case study

#	Characteristic
1	Phenomenon is examined in a natural setting.
2	Data are collected by multiple means.
3	One or few entities are examined.
4	The complexity of the unit is studied intensively.
5	Case studies are more suitable for the exploration, classification and hypothesis development stages of the knowledge building process; the investigator should have a receptive attitude towards exploration.
6	No experimental controls or manipulation are involved.
7	The investigator need not specify the set of independent and dependent variables in advance.
8	The results derived depend heavily on the integrative powers of the investigator.
9	Changes in site selection and data collection methods could take place as the investigator develops new hypotheses.
10	Case research is useful in the study of 'why' and 'how' questions because these deal with operational links to be traced over time rather than with frequency of incidence.
11	The focus is on contemporary events.

Benbaset *et al.*'s characterisation of a case study clearly indicates the appropriateness of the case study research strategy for investigating the objectives presented in chapter two i.e.:

- A case study allows the exploration of previously unexplored aspects of a phenomenon. (Yin [140] argues that all three of the research strategies may be used for exploratory studies.)
- A case study is ideally suited to the study of the actual behaviour of a phenomenon, because the phenomenon is examined within its natural setting.
- A case study is oriented toward the intensive investigation of the complexity of the phenomenon and so is ideally suited to developing an "intimate connection with reality" ([39], p. 532). Accordingly, a case study should provide a solid empirical foundation upon which subsequent investigations may develop theory.
- A case study may be used to replicate the investigations of another study. Kelly and McGrath ([59]), for example, argue that multiple research methods should be used to examine a phenomenon, because the strengths of each method compensate for the weaknesses inherent in the other methods.

An inherent weakness with case studies is the difficulty in generalising their findings to a wider set of cases. This weakness is addressed in the next section and again in chapter nine.

3.3 Heuristics for the design and conduct of case studies

A number of papers have been reviewed in order to establish heuristics for the design and conduct of case studies (i.e. [7, 24, 39, 42, 66, 86, 94, 131]; a number of personal communications with researchers were also conducted i.e. [23, 43, 44, 67, 81, 86]). The heuristics derived from these papers were organised into three sets: advice on the number of cases to use, advice on the design of the case study, and advice on the building of theory. The heuristics are presented in Tables 3.3.1 through 3.3.3.

Cases from a single site strengthen the internal validity of a theory, but one would prefer cases from several sites in order to strengthen the applicability of the theory to a broader set of cases. Heuristic #17 indicates that theories built from case study research are essentially theories of particular types of situation. Thus, the complexity of the phenomenon being observed, the limitations on the number of cases investigated, and the variety of sites from which these cases are drawn all influence the degree to which one can explain the behaviour of a wider set of cases.

Exponents of case-based research methodologies (e.g. [39, 140]) argue that generalisation is not based on the findings of the case study but rather on the theory for which the case is an empirical example. One first demonstrates that the empirical evidence validates the theory for a particular situation and then, through the use of replicated studies, that the theory applies to other, specific settings. (This is why theories built from case study research are essentially theories of particular types of situation.)

This investigation is not seeking to generate or test a *theory*, but it is seeking to provide some degree of explanation. Because this investigation is not generating or testing a theory, one might argue that not only is the investigation free from any obligation to consider generalisability but would be inherently *incapable* of offering any findings that do generalise (because, without a theory, there is no basis on which to claim generalisations, and empirical generalisations are not possible with such a small sample of projects). The conceptual models developed through this investigation do, however, provide a basis on which some generalisation may be made. Furthermore, attention in chapter nine is re-directed back to previous studies in an effort (which is partially successful) to identify other empirical studies whose findings complement those generated in this investigation. Therefore, although a formal theory is not exploited in this investigation, conceptual structures are exploited and do provide some basis on which generalisations may be made.

Table 3.3.1 Heuristics for the number of cases to use

#	Heuristic	Reference(s)	Comments
1	While there is no ideal number of cases from which to build a theory, a number between four and 10 usually works well.	Eisenhardt [39]	With fewer than four cases it is often difficult to generate theory with much complexity, and its empirical grounding is likely to be unconvincing. With more than 10 cases, it quickly becomes difficult to cope with the complexity and volume of the data.
2	Use two cases.	Orlikowski [86, 87]	
3	Single cases are most suitable for exploration, or in the final stages of theory-testing	Benbaset [7]	
4	Multiple cases are preferable for description, theory-building and theory-testing, because they allow cross-case analysis and the extension of theory, and they yield more general research results.	Benbaset [7] Yin [140]	
5	Be clear about why each case is chosen: about what contribution each case is expected to make to the building and testing of the theory.	Cavaye [23]	
6	Cases from a single site (a homogeneous environment) strengthen the internal validity of the theory	Benbaset [7]	
7	If the phenomena to be observed have to be contained within a single or relatively small number of cases then choose cases where the progress is transparently observable.	Pettigrew [94]	Pettigrew also advises: a) Go for extreme situations, critical incidents and social dramas. b) Go for polar types. c) Go for high experience levels of the phenomena under study. d) Go for more informed choice of sites and increase the probability of negotiating access.

Table 3.3.2 Heuristics for designing the method to be used

#	Heuristic	Reference(s)	Comments
8	A case study should adhere to rules of procedure, just as an experiment or a survey are expected to.	Benbaset [7]	
9	Provide sufficient detail on the data collection and analysis methodology, so that it is clear how you came to your findings, and so that others can repeat the study.	Benbaset [7]	Replication needn't be literal. It may be theoretical.
10	Be clear about the research objectives and how these 'translate' into goals, or aims, for the empirical studies (the research questions)	Benbaset [7]	
11	State the units of analysis, and why you chose those units.	Benbaset [7]	Units of analysis dictate the kinds of generalisations than can be made, and the kinds of data that is collected.
12	Specify the interview questions.	Benbaset [7]	
13	Interview different people within each case, in order to gather different perspectives.	Benbaset [7]	This will support triangulation in the analysis stages of the studies.
14	Be clear about the practical constraints on the research.	Pettigrew [94]	For example, time sets a frame of reference for what changes are seen and how these changes are explained.
15	Be explicit about the output of the research.	Pettigrew [94]	For example, be explicit about the varieties of research output, the sequences of this output, and the audiences for this output.
16	Collect data that is processual, comparative, pluralist, historical, and contextual, and collect this data at different levels of analysis.	Pettigrew [94]	

Table 3.3.3 Heuristics for building theory

#	Heuristic	Reference(s)	Comments
17	Theories built from case studies "are essentially theories about specific phenomena"	Eisenhardt [39]	Eisenhardt's own work in this area generated a 'mid-range theory' (a term and concept first introduced by Merton [77])
18	If one is building theory then offer multiple competing theories.	Lee ([66])	Through forwarding a number of theories, we can strengthen our research because the competing theories indicate different interpretations, and highlight threats to particular theories (weakness in the predictions)
19	Be clear about the relationship of the emerging theory to existing theory.	Eisenhardt [39]	Be clear about this relationship before, during, and after the design and conduct of the empirical studies.
20	Given that theory-building using case research is a bottom-up approach, be clear about the kinds of generalisation one intends to make, and the techniques one intends to use to make these generalisations.	Eisenhardt [39]	
21	Prepare for closure.	Eisenhardt [39]	
22	Iterate toward a theory that closely fits the data.	Eisenhardt [39]	
23	Clearly describe the data sources and the way they contribute to the findings of the research.	Benbaset [7]	This is an important aspect of strengthening (clarifying) the reliability and validity of the findings.
24	Exploit contradiction.	Eisenhardt [39]	

With regards to the selection of cases, Pettigrew ([94]) suggests that one chooses polar cases, extreme situations, critical incidents, and cases with high 'experience levels' (see heuristic #7). In choosing such cases, however, one may select cases that are actually quite unusual and therefore not particularly representative, consequently affecting one's ability to generalise with the case.

Pettigrew also recognises that practical constraints limit the number of cases that can be observed, the sites from which these cases can be drawn, and the time-frame within which the cases can be investigated. For the current investigation, there is an approximately twelve-month time-frame to conduct the evidence-gathering portion of the investigation, and a further twelve-month time-frame to complete the analysis of the evidence. Selecting projects that start and complete within the twelve-months of evidence-gathering places a constraint on the kinds of project that can be investigated. This will affect the potential applicability of a subsequent theory i.e. that the theory is applicable to relatively short projects.

Fenton ([40]) provides some cautionary advice on the duration of an investigation:

> "Sometimes research is designed and measured properly but just isn't carried out long enough... the long-term view led to conclusions very different from the short-term view." ([40]; p. 92)

and:

> "Researchers must take a long-term view of practices that promise to have a profound effect on development and maintenance, especially since the resistance of personnel to new techniques and the problems inherent in making radical changes quickly can mislead those who only take a short-term view." ([40]; p. 93)

Against these cautionary words, twelve-month projects are still of reasonable length and it is not uncommon for commercial software development projects to last for such durations.

3.4 The selection of projects for case studies

Five projects were initially selected for case studies from a candidate set of 16 projects, all taken from IBM Hursley Park. Almost immediately, there were problems gaining regular access to two of these projects, and these projects were dropped as case studies and

replaced by a sixth project. As the evidence collection period progressed, it became increasingly clear that it would be impractical to maintain four case studies (because of the demands of collecting and analysing evidence from four cases), so the number of cases was further reduced to two, here called Project B and Project C. Appendix B1 provides a description of the 16 candidate projects, the criteria for selecting the original five cases, and more detail on the reduction of case studies from four to two.

3.5 Summary of the evidence collected

Table 3.5.1 Summary of evidence collected

Type of evidence	Project B	Project C
Interviews	8	9
Meeting minutes, of which:	51	76
- Project status meetings	49	N/A
- Design/Code/Test status meetings	0	37
- Feature commit and approval meetings	N/A	34
- Senior management meetings	1	5
- Project review (post-mortem)	1	N/A
Researchers records of status meetings	2	N/A
Project schedules	1	2
Projector overheads (from presentations)	1	2
Project documents, of which:	6	7
- Plans	3	1
- Other documents	3	0
Risk assessments	2	2
Project 'contract' (including amendments for Project C)	1	1
Feedback workshop questionnaires	1	2
Total number of 'documents'	73	101

Table 3.5.1 summarises the evidence collected for Projects B and C. As the table indicates, naturally occurring evidence was supplemented by the conduct of interviews and a feedback workshop following the completion of the project. 'N/A' indicates that information was not available from the project. Also, the researcher attended two project status meetings for Project B, with the purpose of evaluating the degree to which the minutes of the status meetings represent the actual content of those meetings. The 'learning curve' required to understand the discussion at the meetings meant that this approach was unfeasible, and consequently it was not pursued. The inability to assess the representativeness of the minutes is recognised as a threat to validity of this investigation.

The primary source of evidence used in the analyses was the minutes of status meetings (project status meetings for Project B and design/code/test status meetings for Project C), and these were supplemented by information from interviews, project schedules and the feedback workshop. The status meetings (whether project or design/code/test) are the

highest-level meetings within the respective projects, occur regularly (typically weekly or fortnightly), are typically attended by representatives from process areas important to the given project (e.g. design/code, test, marketing, finance, support; see below for a clarification of this point) and are a naturally occurring phenomenon (so that the researcher is not intruding on the project).

For Project B, project status meetings appear to typically last between 1.5 and two hours, each producing about ten A4 pages of minutes. At every meeting, the first item on the meeting agenda was a discussion of proposed additional design changes (each design change is a set of requirements) for the project. Each design change was either rejected, accepted or deferred for further investigation. The representatives of each process area then reported on the progress of their area. Action Items were also recorded and their progress monitored at each meeting. Overall, the minutes appeared to be structured around the issues concerning the project.

For Project C, design/code/test status meetings appear to typically last between one and 1.5 hours, each producing about six A4 pages of minutes. The minutes were not structured in a regular format like Project B. Proposals for new features were managed through the feature commit and approval meetings (see Table 3.5.1; the minutes for these meetings were very brief). Action Items and their progress were not recorded in the minutes. Overall, the minutes appeared to be structured chronologically i.e. in the order of the discussions that occurred at the meetings. Some recent research on the structure of meetings, and their associated agendas and minutes ([37]), suggests that meetings, agendas and minutes that are focused around issues, rather than chronologically, have a positive effect on the outcome of a project.

Overall, the status minutes provide a broad view of the project over the duration of the project. Naturally, minutes do not record all that was discussed at a meeting, or even necessarily the most important issues, and such meetings are unlikely to discuss all the issues occurring within the project at the time of the meeting. Consequently, there are at least two levels of simplification with meeting minutes. First, in reporting the progress of a process area, the representative of that process area may simplify the progress of that area. Second, the minutes simplify the discussions that occurred at the meeting. Despite these simplifications, the minutes provide a large volume of 'rich' information about the project over the duration of the project, and this evidence appears rich enough to provide a substantive, longitudinal view of the software development process. Furthermore, the minutes provide a level of detail that is unlikely to be collected from other sources of evidence. Conversely, these other sources provide useful insights that are not provided by the minutes.

One potential problem with analysing the minutes of the status meetings is that, as already noted, Project B holds project status meetings, whilst Project C holds only design/code/test status meetings. Project B is a larger project (see chapter five for more information) with a larger number of distinct process areas, and representatives for each of these areas. By contrast, Project C is a small project, with fewer distinct process areas. The design/code/test meetings are attended by members of the design/code and test process areas. It is likely that the content of the design/code/test meetings will be different than the content of project status meetings. For the two projects, these two types of meetings are still the highest-level meetings within the project. Status meetings do not occur for every week of Project B or Project C. Section 6.2.2 presents more information on the frequency of the status meetings for the two projects.

Interestingly, the minutes for both projects do not record any explicit comparisons between the actual progress of the work and the planned progress, as represented in the schedule and the work breakdown structure. It may be that these comparisons are made but not recorded (note, however, that no such discussion occurred at the two meetings attended by the researcher). Another possibility is that the comparisons occurred outside the status meetings (which would be surprising because the status meetings for both projects are an explicit mechanism for reporting the progress of each process area to the rest of the project, and are the highest level meetings in the respective projects).

Table 3.5.2 Summary of interviews

Project	Interview Id.	Week	Role of interviewee
Project B	B.001	8	Project Leader
	B.002	14	Business and Technical Strategy
	B.003	14	Project Leader
	B.004	15	Business and Technical Strategy
	B.005	16	Project Leader
	B.006	17	Lead developer / Project Assistant
	B.007	18	Lead developer / Project Assistant
	B.008	28	Project Leader
Project C	C.001	6	Project Leader
	C.002	8	Project Leader
	C.003	11	Project Assistant
	C.004	13	Brand and Technical Planning
	C.005	13	System Test Manager
	C.006	16	Project Leader
	C.007	25	Project Leader
	C.008	34	Project Leader
	C.009	39	Project Leader

Interviews were open-ended and semi-structured, and the questions prepared for the interviews were dependent on recently analysed evidence (i.e. there was no standard

template of questions). Table 3.5.2 provides a summary of the interviews. A number of interviews were recorded (some interviewees asked not to be recorded) and notes were taken at all interviews. It is important to record interviews in order to aid subsequent analysis and to provide reliable evidence (e.g. verbatim quotes) where these are required to support an argument (personal communications with researchers i.e. [23, 43, 44, 67, 81, 86]). It is not necessary that the recordings of interviews be transcribed (and they were not in this investigation). For reasons of confidentiality, notes from interviews are not included in the appendices to this thesis.

The feedback workshops were conducted approximately one year after the completion of the two projects. For Project B, one workshop was conducted. For Project C, two workshops were conducted (the second workshop addressed outstanding issues from the first). For both projects, the respective Project Leader and Project Assistant were present at the workshops. The workshops took the form of exploring the study's findings with the Project Leader and his assistant, so as to validate and clarify the findings. Van Genuchten ([128]) adopted a similar approach in his study. In this way, the feedback workshops provide one method of validation of the findings from this investigation. The feedback workshops also provided additional information to help clarify and extend the analysis in this investigation.

Conducting the workshops some time after the project's completed was advantageous because project members are likely to have a more objective perspective of their project. Also, with the products in the market for about a year, the project members were able to assess the success of the products. Against these advantages, project members were unable to remember certain information, which meant that certain questions asked during the workshops could not be answered.

3.6 Methods for analysing the evidence

Two methods for analysing the evidence were used in this investigation. The first method is concerned with developing descriptions and explanations of the behaviour of the projects and their schedules. The second method is concerned with replicating part of Bradac *et al.*'s investigation.

A method for developing descriptions and explanations of behaviour

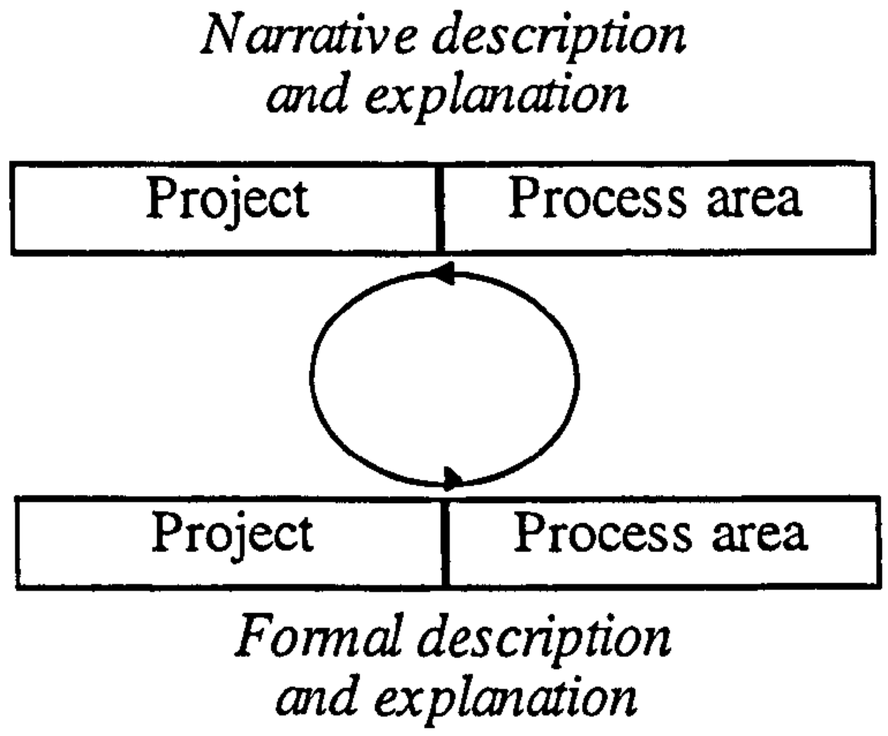


Figure 3.6.1 A method for developing descriptions and explanations of software project behaviour

Descriptions and explanations of behaviour were developed through an iterative two-by-two matrix of analysis, as shown in Figure 3.6.1. More specifically, narrative and formal descriptions and explanations were developed for both the behaviour of the project as a whole and the behaviour of the process areas within the project. The project and the process area are the units of analysis for this set of methods (*cf.* heuristic #11).

This kind of analysis is an intense manual process, which is also iterative and intuitive in nature, requiring the researcher to constantly search and re-search the evidence for particular items of evidence relevant to the respective descriptions and explanations. This process has clear similarities with Benbaset *et al.*'s ([7]) eighth characteristic of a case study (see Table 3.3.1) i.e.

> "The results derived depend heavily on the integrative powers of the investigator." ([7], p. 374)

Table 3.6.1 Summary of the types of analysis conducted

Method	Analysis	Characteristics	Level of project	Evidence	Chapter
One	Narrative description and explanation of the project	Socio-technical contexts. Tactics of management. Actual progress: • Workload • Resource-levels • Schedule • Indicators of project activity • Re-plans • Events • Status information.	Project	Status meeting minutes Interviews Project schedule(s) Project contract(s) Other meeting minutes Feedback workshop(s)	Five
	Narrative description and explanation of process areas	Frequencies, types of, and relationships within: • Waiting • Progress of work • Outstanding work	Process area	Status meeting minutes Interviews	Six, seven & eight
	Formal explanation of the project	Remaining duration. Capability. Workload.	Project	Status meeting minutes Interviews	Four
	Formal explanation of the process areas	Poor progress, outstanding work, waiting	Process area	Status meeting minutes Interviews	Four
	Integration of formal and narrative explanations	Insights common across the above characteristics	Project and process area	Insights from the preceding four analyses	Nine
Two	Testing conjectures from Bradac *et al.*	Prevalence of waiting. Types of waiting.	Process area	Status meeting minutes	Six, seven & eight

This method of analysis satisfies some of the heuristics given in section 3.3 (i.e. heuristics #16 and #23) but finds difficulty satisfying other heuristics given in that section (i.e. heuristics #8 and #10). Benbaset *et al.* add:

> "Using multiple methods of data collection, however, offers the opportunity for triangulation and lends greater support to the researcher's conclusions." ([7], p. 374)

As explained in section 3.5, multiple sources of evidence were used in this investigation, and so lend greater support to the conclusions from this investigation.

The formal explanations (i.e. models), resulting from this analysis, are presented and discussed in chapter four. The narrative descriptions and explanations are presented and discussed in chapters five through eight. The formal and narrative explanations are then integrated in chapter nine. Table 3.6.1 provides more detail on the various analyses that are conducted as part of this investigation.

A method for replicating Bradac *et al.*

Bradac *et al.* ([18]) observed that, for the one designer they studied:

> Waiting is more prevalent during the beginning and end of the project, rather than during the middle of the project.

Three sets of evidence collected from Projects B and C provide an opportunity to test this conjecture. The three sets of evidence are:

- Reports of waiting
- Reports of poor progress
- Reports of outstanding work

Reports of waiting most clearly relate to Bradac *et al.*'s investigation of waiting. Using the model of capability, presented in chapter four, the other two sets of evidence can also be used to test this conjecture.

The method for testing Bradac *et al.*'s conjecture consists of collecting evidence on the frequency of waiting, poor progress and outstanding work per week, for the duration of the project; to organise this evidence into three sets, representing the beginning, middle and end of the project (see section 3.7 for a definition of the beginning, middle and end of

a project); and then to compare (using a Mann Whitney *U* test) the median frequencies of reports for the middle and end of the project. If the median for the end of the project is significantly greater than the median for the middle of the project then the waiting, poor progress or outstanding work is considered to be more prevalent during the end of the project than during the middle of the project (and Bradac *et al.*'s conjecture is confirmed). Mann Whitney *U* tests were chosen because it is not clear that the samples of reporting evidence are drawn from populations with a Normal distribution.

Table 3.6.2 Comparison of Bradac *et al.*'s research design with this investigation

Feature of the research design	Bradac *et al.*'s study	This investigation
Focus of inquiry	Local process (designer)	Process areas
Duration of evidence	30 months	Approx. 12 months per project
Source evidence	Designer's actual behaviour	Minutes of status meetings
Analysed evidence	Designer's recorded behaviour	Evidence extracted from source evidence
Amount of evidence collected that was used	All the evidence	Waiting, poor progress and outstanding work evidence
Application of Bradac *et al.*'s characterisation	Classification applied by designer in 'real-time'	Retrospective classification of evidence by researcher after the completion of the project
Numbers of samples	One sample from one project	Six samples, three from each project

Table 3.6.2 compares the designs of Bradac *et al.*'s study and the current investigation. Differences in results between Bradac *et al.*'s study and the current study may partly be due to differences in the research designs.

3.7 Operational details of the investigation

Organising the evidence relating to process areas

In order to investigate the characteristics of waiting, outstanding work and the progress of work (so as to replicate parts of Bradac *et al.*'s study and to investigate the behaviour of process areas) the minutes of the status meetings were searched, using a text editor, for particular phrases.

Table 3.7.1 Phrases for searching the minutes of status meetings

Evidence	Phrase	Derivatives (examples)
Reports of waiting	wait	waiting, awaiting, await
	block	blocked, blocking
	held up	
	hold	holding (holding up)
Reports of outstanding work	outstanding	
	backlog	
Reports of progress of work	progress	

A number of phrases were acceptable for each set of references. These phrases are presented in Table 3.7.1. Each phrase also 'encapsulates' derivatives (e.g. stemmed words) of that term. There were search options in the text editor[2] allowing a search on an entire word (e.g. select only the term 'wait') or on embedded words (e.g. select such terms as 'await' or 'awaiting'). The terms presented in Table 3.7.1 are not exhaustive, in that they do not contain all the different kinds of terms that could possibly represent references to waiting, outstanding work or progress. The table is complete in that it lists all of the terms that were used in the searches.

As the text editor could search across a series of text files, all of the text files for a project were stored within the same directory, and the search was conducted across all files within that directory. Thus, one search would search all of the evidence for one project. For each project, three searches were conducted, one search for each of the three sets of references.

Upon completion of each search, the text editor presented a list of each occurrence of a term (or a derivative of that term), together with the text file within which that term occurred. If there was more than one occurrence in a text file, the text editor would list each of the occurrences of the term. This produced an initial set of all references. This initial set was then refined based on three criteria:

1. Whether the term was in an appropriate context. For example, sometimes occurrences of the term 'block' referred to design issues (e.g. a STATE block) rather than process issues. Such occurrences were removed from the set.
2. Whether there were duplicate terms within the same 'chunk of meaning' (e.g. a sentence). For example, the phrase "work is held up because we are waiting on a fix" would be selected twice by the text editor. These duplicate references were removed.

---

[2] The text editor that was used was *BBEdit Lite* version 4.1 from Bare Bones software (http://web.barebones.com)

3. Whether the term was identified within the context of an action item. For Project B, Action items were recorded twice in the minutes of a meeting: first, at the 'point' in the minutes where the action item was raised; second, in a separate summary at the end of the minutes, where all action items (opened in the meeting, outstanding from previous meetings and closed in the meeting) are recorded. Duplicate references of this sort were also removed from the list.

Having refined the set of references, each of these references (together with their surrounding 'chunk of meaning') was then copied into a separate text file and labelled with the week number in which it occurred. Each item was then classified in various ways (see chapters six through eight for more information on the classifications).

Additional analyses for the waiting evidence

With regards to the types of work on which a process area was waiting, two classifications were used. The first classification was 'inductive' in that the types were first identified from the items of evidence for each project, and then aggregated across the two projects (so as to form a common classification system across the two projects). This first classification was then mapped to Bradac *et al.*'s classification so that parts of Bradac *et al.*'s study could be investigated. Appendix B3 provides information on the first classification and their mappings to Bradac *et al.*'s classification. There were two reasons for using two classifications. First, Bradac *et al.*'s classification may not be a useful classification system for the evidence collected in this study. Second, the first classification may provide opportunities for insights into the 'Waiting on other' category of Bradac *et al.*'s classification.

Adjusting Bradac *et al.*'s waiting evidence

Because this study is only investigating references to waiting, whilst Bradac *et al.* studied observations of working and waiting, some adjustments need to be made to their percentages of time spent waiting. Rather than using percentages of types of waiting relative to the total time (i.e. waiting time and working time), the evidence was adjusted so that the analysis uses percentages of types of waiting relative only to the time spent waiting (i.e. excluding working time).

Table 3.7.2 Adjusted percentages from Bradac *et al.*'s study

State	% total time	% waiting
Waiting on the laboratory	2.7	4.5
Waiting on an expert	3.1	5.1
Waiting on a review	9.2	15.1
Waiting on hardware	1.0	1.6
Waiting on software	1.9	3.1
Waiting on documentation	2.4	3.9
Waiting on other (also known as Other)	40.7	66.7
Total	61.0	100.0

Table 3.7.2 summaries the adjustments to the percentages of time spent waiting. The middle column presents the original percentages. The column on the right presents the adjustments to the percentages. With the adjusted percentages of waiting it now becomes clear that almost 67% of the time spent waiting was spent in the 'Waiting on other' state.

Mapping phases of Projects B and C to Bradac *et al.*'s study

As explained in chapter two, Bradac *et al.* identified several tasks that the designer they studied might be doing. In observing that waiting is more prevalent during the beginning and end of a project, rather than during the middle of a project, Bradac *et al.* appear to map their tasks to the beginning, middle and end of a project. Bradac *et al.* do not, however, clearly define which tasks mapped to which phase of the project.

Table 3.7.3 'Mapping' Bradac *et al.*'s tasks to the phases of Projects B and C

Tasks in Bradac *et al.*'s study	Part of the project	Phases of Projects B and C
Estimate and Investigate	Beginning	Plan
Plan Development	Beginning	
Requirements	Beginning	
High Level Design	Beginning	
Low Level Design	Middle	Design/Code
Write Test Plans	Middle	Functional verification
Code	Middle	
Inspections and Walk-throughs	Middle	
Low Level Test	Middle	
High Level Test	End	System test
Customer Documentation	End	
Support	End	
Project Retrospect	End	

Table 3.7.3 presents the tasks identified in Bradac *et al.*'s study, together with an interpretation of which tasks, in Bradac *et al.*'s study, and which phases, of Projects B and C, 'map' to the beginning, middle or end of the project. The table indicates that, for

Projects B and C, the plan phase maps to the beginning of the project, the design/code and functional verification phases map to the middle of the project, and the system test phase maps to the end of the project. Although the plan phase maps to the beginning of the project, this is not to say that *planning* does not occur throughout the duration of the project. For example, Rook writes:

> "While the major effort on planning is required during the project initiation phase, planning continues from phase to phase, as further details become apparent, and as changes are introduced." (see [75], chapter 27 page 19)

Chapter 4 Three analytic models

4.1 Introduction

During the collection and preliminary analysis of the evidence from Projects B and C, two models were developed to help subsequently organise and analyse the evidence. The first model, a simple model of software project schedule behaviour, is used to describe and analyse characteristics of the project, at the level of the project. This model is used primarily in chapters five and nine. The second model, a model of capability, is used to describe and analyse characteristics of the process areas within the project. This model is used primarily in chapters six through nine. The two models have been integrated into a third model, the integrated model of schedule behaviour and capability. This model is used in chapter nine.

4.2 A simple model of software project schedule behaviour

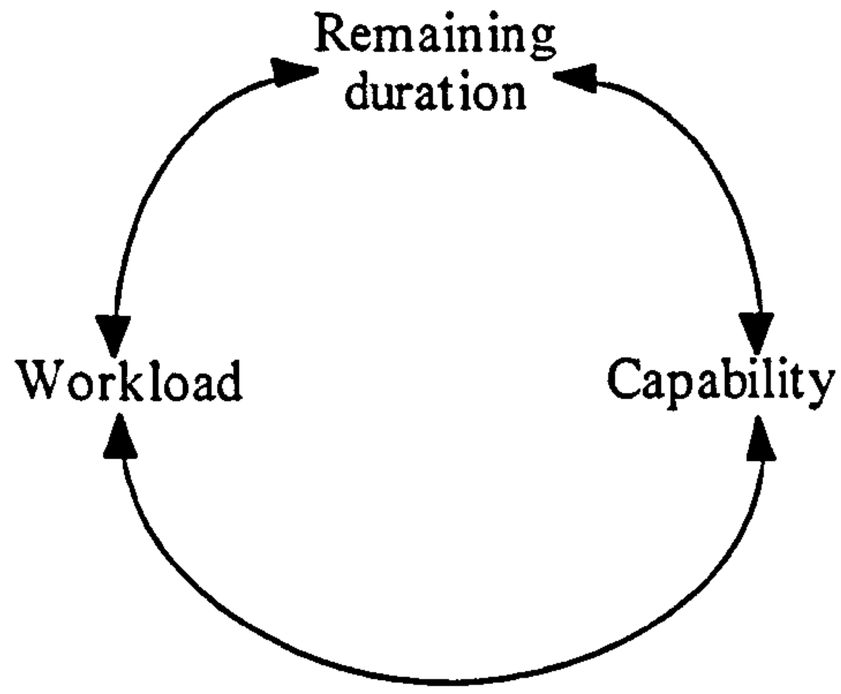


Figure 4.2.1 A simple model of software project schedule behaviour

Figure 4.2.1 presents a simple model of software project schedule behaviour, consisting of relationships between three constructs. Remaining-duration is defined as the period of time for which the remainder of the project will last, at time *t* of the project. Workload is defined as the number of units of work remaining to be completed, at time *t* of the project. (There are a number of potential measures of units of work, e.g. lines of code, modules, function points, features etc., and there are benefits, such as triangulation, in exploiting these different measures.) Capability is defined as the ability to complete *n* units of work per unit time, at time *t* of the project. Capability incorporates concepts of productivity and resource.

The logic of the model is that a change in one of the constructs will affect a change in one or both of the other constructs. The relationships of particular interest to the current investigation are:

- An increase in workload will lead to a proportional increase in remaining-duration, unless there is a proportional increase in capability. Examples of an increase in workload are the introduction of new requirements and rework.
- A decrease in capability will lead to a proportional increase in remaining-duration, unless there is a proportional decrease in workload. An example of a decrease in capability is skilled personnel leaving the project.

The stability of the project's schedule is dependent on the 'balancing' of the project's capability and workload. Within the context of this model, the difficulty in managing projects is recognising what changes need to be made to capability or workload in order to maintain the stability of the schedule.

In principle, the model can be applied to various aspects of the project e.g. to the project as a whole, to a particular process area such as design, or to a particular part of the product such as a feature. Three of these aspects are explored in chapter five, but the applicability of the model to various aspects of a project still requires further investigation. The development of the model has been documented elsewhere ([99, 100]).

Examples of the logic of the model

Two brief examples of the logic of the model are examined. In the first example, workload increases. In the second example, capability reduces.

1. Consider a project with 12 units of work and a planned project duration of 12 months. The *mean* capability for the project is one unit per month. If, after the end of six months, an additional unit of work is added to the project (for example, through new requirements, rework or undiscovered work) then the project has 7 units of workload, and will need to increase its mean capability to $^{7}/_{6}^{th}$ (i.e. 1·167) units per month to complete the work in six months time.

2. Consider, again, a project with 12 units of work and a planned project duration of 12 months. The *mean* capability for the project is, again, one unit per month. If, after the end of six months, capability reduces by $^{1}/_{6}^{th}$ (i.e. 0·167) units per month (for example, because of the departure of personnel to another project) then the project

would need to reduce its workload from six units to five units in order to complete the work in six months time.

Support for the model

Some support for the model of schedule behaviour is available from previous research. Olsen ([83]) distinguishes between change demand (comparable to workload) and change service (comparable to capability) and uses a theoretical metric, the change point, as a measure of change demand rate and change service rate. Schriber and Gutek ([107]) define pace, a concept similar to capability, as:

> "... the rate at which activities can be accomplished (i.e. the speed of activity or the number of activities that can be done within a given deadline)." ([107]; p. 643).

Blackburn *et al.* ([10]) distinguish between development speed and productivity, and they argue:

> "Development speed and productivity are not the same because low productivity organizations can be quicker to market by throwing human resource - armies of programmers - at the project." ([10], p. 876)

For two projects with the same workload that complete within the same duration, but one that is more productive and one that has more resource, both projects have the same capability. (It is likely that the more productive project will incur less costs.)

Rook's (see [75]) definition of a work breakdown structure is also closely related to the concept of workload. McDermid writes:

> "The work breakdown structure (WBS) is a product-oriented task hierarchy of all the work to be performed to accomplish the project contractual objectives. The products may be elements of software, hardware, documents, tests, reports, support services, or other quantified elements of the objectives." ([75], chapter 27 page 20)

As noted by McDermid, a work breakdown structure identifies all *quantified* elements of the project's *contractual* objectives. In contrast to a work breakdown structure, the concept of workload incorporates qualitative elements and non-contractual objectives (which still introduce work into the project). The concept of workload also incorporates

invisible work. Nardi and Engestrom ([82]) edit a special issue of the journal *Computer Supported Cooperative Work* that investigates the nature and structure of invisible work. They write:

> "... invisible work takes many guises: as tacit and contextual knowledge, as informal social networks, as expertise acquired by old hands, as long-term teamwork." ([82], p. 2)

The model also finds some implicit support from project managers at IBM Hursley Park. For example, the Project Leader of Project A states:

> 'First determine the work to be done; then determine our ability to do that work; then build a plan from these.' [Interview A.008.AR]

Caveats to the model

As already explained, the model was developed during the collection and preliminary analysis of the evidence. This has both advantages and disadvantages. Yin ([140]), for example, would disagree with this approach, arguing that the model should be developed in some form prior to the collection of the evidence. By contrast, Strauss and Corbin ([120]), as another example, would favour the general approach taken here, but they might *disagree* with the specific approach, arguing that it is not sufficiently 'grounded' in the evidence. These two examples indicate that the method by which the model was developed is a source for debate. Subsequent analysis and discussion (see chapters five through nine) indicate that the model is useful for describing, organising, explaining and communicating the behaviour of Projects B and C. The important issue is to recognise the model as a conceptual tool with both strengths and weaknesses, and with opportunities and requirements for subsequent development and validation.

Distinct from the methodological concerns, practical constraints meant that evidence had to be collected (because the projects had started) before *a priori* models could be fully developed (*cf.* heuristic #14 in Table 3.3.2). Also, it was considered important to develop a model to which practitioners could relate, because this would increase the likelihood that the model would reflect empirical reality, and be useful to practitioners. This necessarily requires that one collect and analyse evidence before developing a model.

Although the logic of the model relates the changes in one construct to the changes in the other two constructs, there is no explicit recognition of how a construct would change in the first place. Other, sometimes more subtle, processes are assumed to cause an initial

change. The model of capability identifies some of the subtler processes for the capability construct.

Whilst the model recognises relationships between workload, capability and remaining-duration, it is not able to distinguish the degrees of change within each construct. Consequently, proportional changes between constructs cannot be assessed. In one respect this is accepted as a limitation of the model imposed by the kinds of evidence that are naturally available from the project. This limitation is overcome, to some degree, by the collection and analyses of various sources of evidence from the project, such as summary status reports of the progress of features. In another respect, however, a model that only represents precise and specific changes would exclude much, if not most, of the qualitative evidence that has been collected. This is undesirable. Consequently, a degree of rigour is sacrificed in the model to improve its utility. In this way, the model is more tolerant of the qualitative evidence and, consequently, the volume and content of the qualitative evidence can be better exploited.

Finally, no distinction is made, at this stage, between the actual, desired, planned and perceived values of remaining-duration, workload, and capability. As indicated above, and within the context of the model, the difficulty in managing projects is recognising what changes need to be made to capability or workload in order to maintain the stability of the schedule. This is a 'conflict' between the planned, actual, desired and perceived values of the three constructs.

Problems with the model

The concepts represented in the model, in particular workload and capability, and the relationships between these concepts are extremely difficult to formalise effectively. For example, certain events in a project (such as the automation of a task) may be treated as a reduction in workload or an increase in capability. Also, there are many different types of work. Design work, considered an intellectually intensive task, appears to be very different from controlling a test library, which is considered a clerical task (*cf.* [46]). A common *measure* of the workload involved with different types of task appears to be impossible to define (which is presumably why Olsen settled for a theoretical metric). Such problems do not prevent the investigation of these constructs, but they do limit the kinds of insights that one can derive from such investigations. For example, because of the difficulty in formalising these constructs, reliable prediction systems are extremely difficult to develop.

4.3 A model of capability

In addition to the model of software project schedule behaviour, a second model emerged from the preliminary analysis of the evidence. This model represents three constructs relating to capability. The model is an attempt to relate previous research reviewed in chapter two to the evidence collected from Projects B and C.

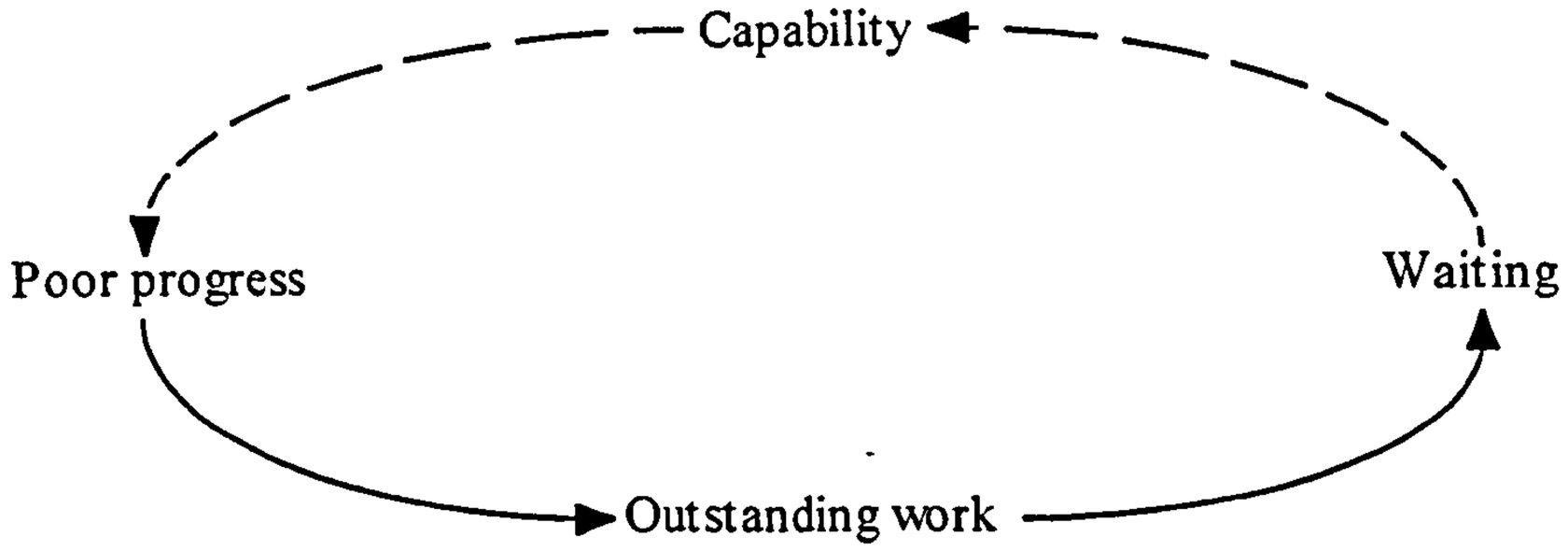


Figure 4.3.1 A model of capability

Figure 4.3.1 presents the model of capability. The main relationships are shown with solid lines. The broken lines suggest possible relationships. The model has the following logic:

1. Two types of imbalance between workload and capability (workload is not shown in Figure 4.3.1) lead to the poor progress of work. The two types of imbalance are those identified in section 4.2 i.e.
   - An increase in workload without a proportional increase in capability.
   - A decrease in capability without a proportional decrease in workload.

2. Poor progress leads to outstanding work.

3. Outstanding work leads to waiting elsewhere in the project (either within the same process area or in another process area). This is because some output has not been produced when it was planned or because resource was reassigned. With the model, waiting points to *subsequent* threats to capability, and reflects preceding imbalances between workload and capability.

4. Because another part of the project has not received an input (or resource) when planned, it must wait on that input (or resource). The waiting threatens the capability of that other part of the project i.e. there is the potential for a reduction in capability because another part of the project is unable to progress.

5. Lower *actual* capability leads to an imbalance between workload and capability. This then causes poor progress, and the logic returns to point 1 above. (See section 4.4 for a discussion of the circularity of the logic of the model.)

4.4 Integrating the models

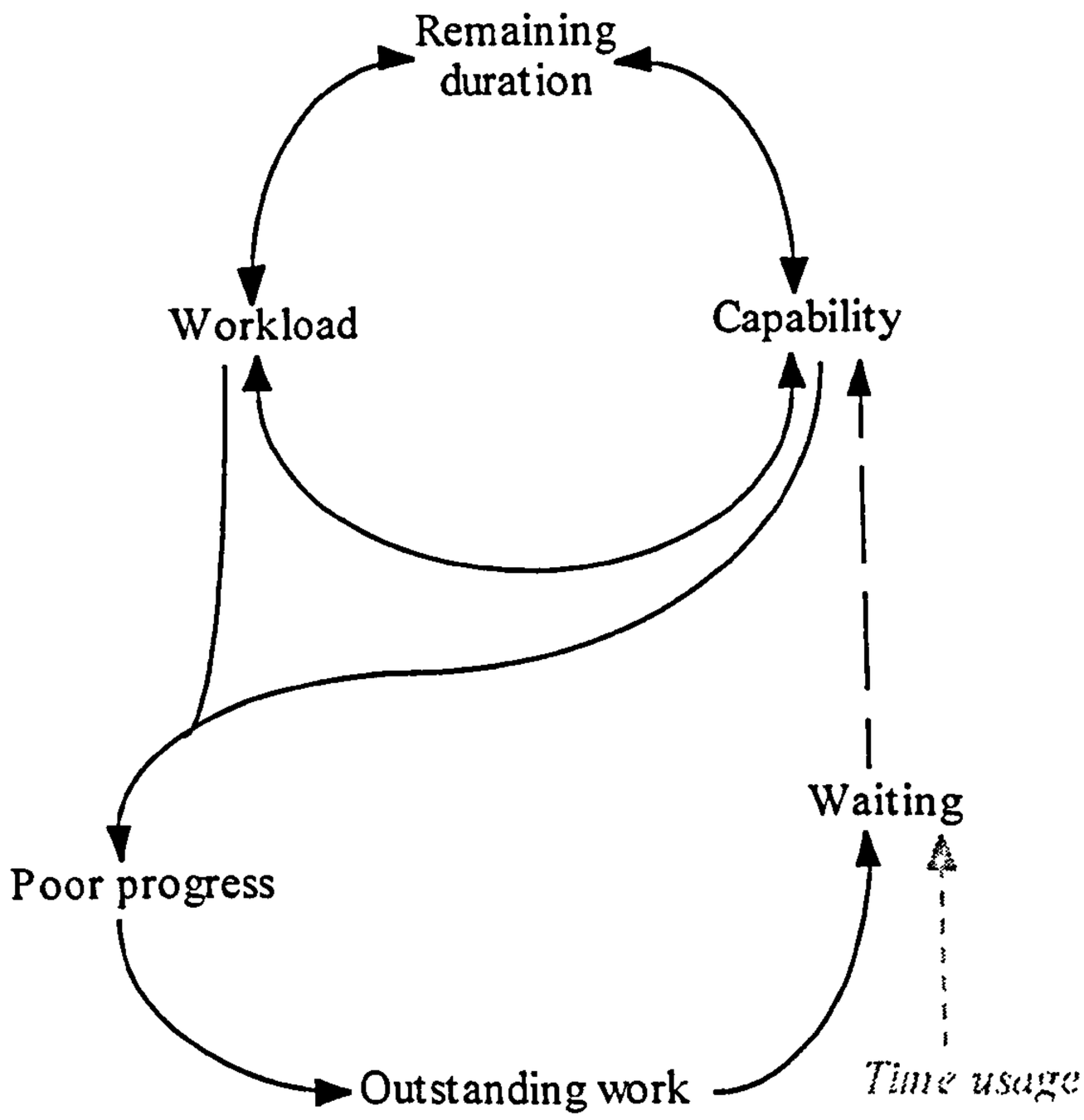


Figure 4.4.1 An integrated model of schedule behaviour and capability

Figure 4.4.1 integrates the model of software project schedule behaviour with the model of capability. It also indicates where studies of actual time usage (reviewed in chapter two) are relevant. In the figure, the integrated model explicitly indicates that poor progress is a function of workload and capability.

The integrated model partially satisfies one of the objectives of this inquiry (see chapter two), by relating lower-level processes (i.e. relationships within and between process areas) with higher-level processes (i.e. relationships at the project-level), and through this integration suggesting the possible effects of lower-level processes on software project schedule behaviour. The objective is only partially satisfied, however, because these models have yet to be formally validated.

With the integration of the model of capability and the model of schedule behaviour, the constructs of the model of capability must also be defined dynamically i.e. poor progress becomes poor progress occurring at time *t* in the project, outstanding work becomes

outstanding work occurring at time *t* in the project, and waiting becomes waiting occurring at time *t* in the project.

The effect of waiting may also be influenced by the level of the process and hence the granularity of the work. Bradac *et al.* focused on a designer waiting on, for example, designs to be delivered from the library. In such an example, the units of work cannot easily be further divided. By contrast, chapter five shows that when the design/code phase actually completes late, both projects start their test phases when planned. At the phase-level, work can be further divided: some of the design/code work will have been completed and this can be passed to the test phase. Phrased another way, work at the phase-level is not discrete in the way that work at the individual level is. This will affect the impact of waiting, as Bradac *et al.* recognised with their requirement for the global process to be 'consonant' with the local process.

It is clear from Figure 4.4.1 (but also Figures 4.2.1 and 4.3.1) that the model consists of two sets of circular relationships: one involving the workload, capability and remaining-duration constructs; the other involving the poor progress, outstanding work and waiting constructs. These circular relationships may be modelled as feedback systems in system dynamic models (e.g. [2, 41]). The feedback relationships may have delays between the cause and effect. Modelling the feedback relationships is beyond the scope of this thesis and stands as an opportunity for further research.

4.5 Alternative models

Some attention was directed at the development of alternative models of schedule behaviour and capability. These included mathematical models using differential equations, system dynamics models (which also involve, at their core, differential equations) and queueing models. While all of these types of models are interesting, and may provide valuable insights, the rigour of these models means that they would demand types of evidence (i.e. well-defined, quantitative evidence) that is not readily available from Projects B and C. This relates back to a point made earlier i.e. that a certain amount of rigour is sacrificed to improve utility.

Also, some of the constructs presented in the model of schedule behaviour and the model of capability may be defined differently. In particular, the 'outstanding work' construct may not just refer to work that should have been completed but hasn't been completed, but may also refer to work that is *yet to be done*. As a manager approaches a deadline, they may consider *all* of the work that they have left to do, some of which may be work that should have been completed by that stage, and some of which is work that was

planned to be completed in the remaining period leading up to the deadline. With this definition, references to outstanding work become indicators of a manager evaluating their ability (and probability) to achieve their goal. Similarly, the poor progress construct may be more directly related to only capability, rather than a ratio of workload and capability.

These alternative models and definitions of constructs reflect the complexity of the phenomena being observed and the difficulty in properly representing that complexity. The alternative models and definitions stand as opportunities for further research.

4.6 Summary

Two separate models have been developed to help organise, analyse and communicate the behaviour of Projects B and C. These models have also been integrated, in order to show how lower-level processes might affect higher-level processes and the schedule behaviour of a project. The two models were related to previous research, and they will be used to explain behaviour at the level of the project and the level of process areas.

Chapter 5 Project-level behaviour

5.1 Introduction

This chapter describes and explains the project-level behaviour of Projects B and C. The model of software project schedule behaviour is used as a basis for these descriptions and explanations, and is applied from three different perspectives:

- The socio-technical contexts of each project (i.e. considering the social/organisational and technical issues, and how these issues interact).
- The actual progress of each project.
- The tactics used to manage each project.

Chapter nine relates the analyses presented in this chapter with the analyses presented in chapters six through eight.

The figures presented in this chapter attempt to efficiently communicate a large volume of qualitative and quantitative evidence from a variety of different sources. The figures are based on Miles and Huberman's ([79]) advice to organise multiple sources and types of qualitative evidence according to time. Also, the visualisation of evidence from a number of different sources (by placing that evidence within the same figure) may reveal subtle relationships between aspects of a project ([124, 125]). A complete explanation of the structure and notation used in the figures is presented in Appendix B2.

5.2 The socio-technical contexts of Projects B and C

Project B

Project B is one release of a middleware transaction processing system (here known as Product B) that operates on mainframe computers. Other versions within the 'family' operate on mid-range machines and workstations. The release preceding Project B, release B-1, introduced new transaction logging functionality that required specific hardware to operate; hardware that was not commonly used by customers. Project B-1 was also a re-packaging of the middleware product with a systems management product (here known as Product BS) that manages the concurrent operation of multiple instances of the middleware product.

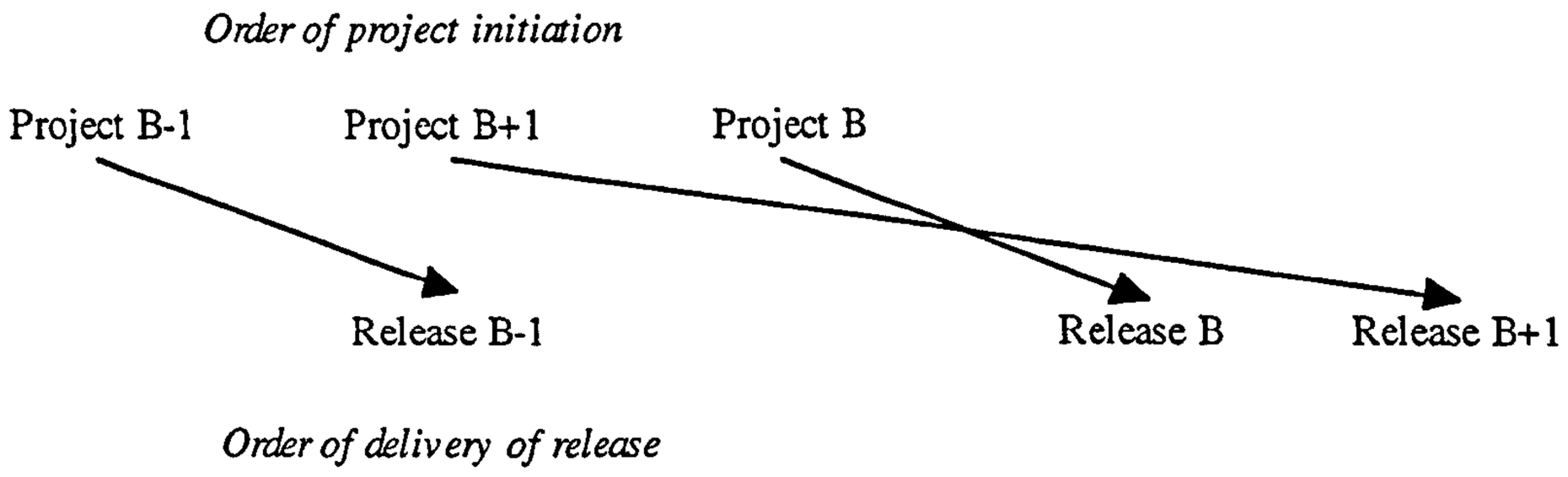


Figure 5.2.1 The relationship between three releases of Product B

The product area recognised that the requirements of specific hardware for transaction logging restricted the product's market, and they needed to correct this issue quickly. Because major releases of the product typically occur in a rhythmic cycle of approximately 18 to 24 months (*cf.* [20]), a minor release was required to deliver a software alternative to the hardware-based functionality. The primary purpose of Project B was to deliver this software alternative. In addition, Project B also provided an opportunity to deliver some functionality that should have been delivered in release B-1 and some functionality that was planned to be delivered in release B+1. Note that Project B+1 actually started before Project B. Figure 5.2.1 illustrates the relationships between the three projects.

Project B recognised that it was more effective for the software transaction logging functionality to be provided via the operating system rather than within Product B itself. This was because the functionality would be more efficient to develop, but also because the product would perform more efficiently when in operation. The operating system is maintained and developed by a product area external to IBM Hursley Park but within the corporation. The external product area designed and coded the transaction logging function and Project B tested it. Project B is also one of four successive projects which are costed as a group. This arrangement might affect the planned staff levels for Project B.

Overall, Project B was considered a success and, as one criterion of this success, the release was delivered when originally planned. Closer inspection of the project indicates that at two features were not delivered with the product, and that the quality of one of these features was lower than desired when it was finally delivered to the market (via the World Wide Web) some weeks later.

Certain elements of the project's socio-technical context clearly relate to workload and capability. With regards to workload, the strategy adopted by the Project Leader was to

limit the changes that might occur on the project. The Project Leader, 'BM', described the strategy he adopted for managing the project:

> "... my stance is that I'm accepting no ... [design changes] - those are the words I use. Minimum change on this project is the most important thing. So, for example I'm running a... [Defect Screen Team]... from day one. And the first topic of my weekly status meetings is [design changes], where I reject them all..." [Interview B.001.BM]

And, in the same tone:

> "When I did the concept I said if anything impacts the base of the code it will be rejected... All the team leaders have done their most to minimise the impact to the base... I'm minimising my risk yet again..." [Interview B.001.BM]

But the Project Leader conceded that:

> "There are some... [design changes] we have to do. But I am accepting no more ... [design changes]." [Interview B.001.BM]

With regards to capability, the structure of the project's management team helps to reduce communication and co-ordination overheads. During the progress of Project B (and Project C), a new set of business processes were introduced across the laboratory. The Project Assistant, 'BF', explained some of the beneficial effects of this new process:

> "[The new business process is]... good because it made individuals more accountable... If people are not accountable, then the project will drift. With [Product B], there is a real knock-on effect. If development slips, then [functional verification] will slip, then system test, etc. [With the new business process], we're more of a team. Barriers are being broken down. Now, strategy, finance, system test etc. - everyone is at the same meeting, working together, communicating with each other, co-ordinating." [Interview B.006.BF]

With the new business process, the project management teams are multi-functional teams comprising representatives from each of the significant process areas of the project. The Project Leader takes a much broader view than any of the particular representatives. The Project Leader gave his interpretation of his role:

> "As a [Project Leader] my role is not just development. I'm concerned with a much broader set of things. [BW] is development... so am I worried about development? No, not really, that's [BW's] problem. I've got enough problems on a grander scale." [Interview B.001.BM]

There is also a close resourcing relationship between Project B and Project B+1 which potentially affects the capability of both projects. A senior member of the project explained:

> "... resource issues cross the boundaries of [Project B] and [Project B+1] because the two projects/products are closely linked. If [Project B] is impacted by resource, then this will affect [Project B+1]." [Interview B.002.BA]

This is consistent with a comment made by the Project Leader:

> "My view is to deliver on a date, so as to release resources for [Project B+1], but also to maintain quality, and provide some functionality." [Interview B.001.BM]

Resourcing clearly leads to the inter-dependence of projects, and indicates how project schedule slippage has affects within the organisation as well as affects on sales etc. to customers. There are similarities here with Perry *et al.*'s ([92]) recognition that designers may not be working on their planned work because they are assigned to a higher priority project (see chapter two for more information).

Project C

Product C is a 'local', cross-platform, middleware transaction processing system that is used primarily in the 'front office' of banks. (By contrast, Product B might be used in the 'back office' of banks. Product C is not the workstation equivalent of Product B.) Product C runs on the DOS, OS/2 and AIX platforms. Project C is an investment to protect the product. The Project Leader, 'CP', explained:

> "What we're trying to propose is the right level of investment that maximises the revenue, and keeps the product going as long as possible." [Interview C.001.CP]

The objective of Project C was to port the existing product to run on a new operating system (which is developed and maintained by another organisation), and to provide some additional functionality for the DOS and OS/2 versions.

Overall, the Project Leader considered Project C to be a success. This was despite the fact that the schedule was re-planned, and that some of the functionality was delivered via the World Wide Web rather than with the product.

Like Project B, there are a number of elements of the socio-technical contexts of Project C that relate to workload, capability and duration. With regards to duration, the Project Leader explained how the product delivery date was determined:

> "[The product delivery date]... is really driven by the 19 person-years effort, to a certain extent... I need to bring on some extra people earlier in the year, so I've got to take them off later in the year to make the 19 person-years fit." [Interview C.001.CP]

This is a clear example of how duration and capability are fixed, with the implication that the workload will need to be determined accordingly. The Project Leader implies such a situation:

> "I would have to say that the planning has been done somewhat backwards here, as we have the schedule and man-power constraints, and we've been trying to fit the work into that, rather than asking people how long it will take them, and building the schedule from that." [Interview C.001.CP]

These constraints then affect the planning process:

> "The basic process was to get the people who would pick those feature up to do the sizings, factor in the service estimates, and then adjust from there to try and make it fit to the [design] phase we thought we would need to meet [our product delivery date]." [C.001.CP]

From Olsen's ([84]) arguments, it would appear that Project C's planning process is common in software projects:

> "In practice, software engineers are often given a fixed deadline and expected to develop a schedule that meets that goal. This fixation on time is not an aberration or the result of misguided management, but the foremost customer requirement and the primary force behind profit. As such time dominates all factors of the software-engineering process." ([84], p. 28/29)

As already noted, the workload will need to be adjusted to balance the capability of the project. The Project Leader explained how the workload looked impossible given the planned schedule, but how he justified that the work could be done:

> "And its actually frightening if you look at... [the workload]... in terms of the productivity that's needed to get this product out of the door. However, the counter argument is that there is very little new function. If you look at the lines of code for... [the new product]... its something like 55 KLOC, and I'm trying to do that with three person years, which looks impossible. However, that is reusing code, its porting code. Where we're writing new code its usually with existing design, where the architecture is already there. I can justify it to myself that its do-able... Its not writing new code, its not using old code without change, its somewhere between those two." [Interview C.001.CP]

The funding constraints, mentioned above, constrain the capability of the project. The Project Leader explained:

> "So we had a resource funding constraint.... I asked can we spend any more on development, and the answer was very much no. This 19 person years is fixed... But its a good business case. Even if we don't develop... [a version of the product to run on a new operating system]..., then I still have to spend 15 person years on service." [Interview C.001.CP]

This is an example of a factor, cost, that is not modelled in the model of schedule behaviour but does affect the constructs within the model (see chapter four for more information).

The resource constraints also affect the organisation of the new development and support teams. Unlike Project B, Project C has a combined development and support team. The Project Leader explained:

> "In an ideal world, one would have... separate... [support]... and development teams, but this would probably be inefficient... You have to remember we've got 19 people here and we're trying to support three products, not one product, and we're trying to develop a new product. And we're actually trying to do an awful lot with very little resource." [Interview C.001.CP]

The combined team means that each person has development and support responsibilities:

> "...each developer has responsibility for some number of components in the product. So its not a case of having three people doing development for the... [new]... product. Each person will have a mix of... [support]... and development responsibility, so it really depends..." [Interview C.001.CP]

> "There's about 9 people who have responsibilities for... [the new product]... but that obviously is not their full-time job. We plan for the service work, which comes in fits and starts. A high severity problem can take a person out for a month. The... [support]... take priority to the development work.. " [Interview C.001.CP]

The funding constraints also affect the composition of the testing team. The Project Leader explained some of the risks for Project C:

> "I can see high risk areas. I've already mentioned service workload. System test is fairly high risk, because of the resource constraints... We've only got funding for two system testers for six months which isn't sufficient to do a good job basically. They're starting later than I want them to start. So already I'm trying to get extra help. Someone from marketing and some people from... [support]. So I'm trying to sort that out 'through back door methods' in terms of getting some extra help for free. Cos [sic] I can't guarantee three system testers." [Interview C.001.CP]

and:

> "A major constraint is actually can we get them [the features] tested, rather than can we develop them. It all boils down to can we get the right skills." [Interview C.001.CP]

The Project Assistant also recognised system test as a concern:

> "Biggest concern is testing. We have one junior person leading an inexperienced test team." [Interview C.003.CG]

Finally, the Project Leader had to fulfil a number of roles (unlike the Project Leader of Project B) and he explained the difficulties these multiple roles might cause:

"One of the problems I have as [Project Leader] and Development manager is finding the time to do both jobs as well as they need to be done. And actually finding the time to devote as much time as I should to the project management side of it is going to be a major challenge for me I think. Because its not only managing the new development stuff, its the... [support]... stuff involved that actually takes a lot of the time... service extensions and all that stuff." [Interview C.001.CP]

This is even more pertinent, given the fact that while the Project Leader has considerable experience in software development, prior to Project C he has not managed an entire project as Project Leader. With the introduction of the new business process, the Project Leader was promoted from design/code manager to Project Leader. He states:

> "I haven't carried anything through the [entire] development process [until now]." [Interview C.007.CP]

Similarities and differences between Projects B and C

In addition to the insights specific to the individual projects presented above, it is also possible to identify a number of characteristics that distinguish and unite the two projects.

Tables 5.2.1 and 5.2.2 present a number of characteristics, and contrast the two projects according to these characteristics. Three entries in Table 5.2.1 require clarification. First, the strategic value of the two products is *relative* to the two products. Although Product C has a lower strategic value this is not to say that the product is not valued by the organisation (if the product had a low value to the organisation it is unlikely it would be maintained). Second, although design changes and additional features are unplanned, this is not to say that such work is unexpected. Experienced Project Leaders recognise that the workload for a project will probably increase. Third, the KLOC sizes of the two projects might misleading suggest that Project C is very much more productive than Project B. Product B is, however, a mission-critical product requiring very high levels of reliability. In addition, much of the code for Product C is being ported from an existing version of the product. The differences between the two products are recognised by Project C's Project Leader:

> "There are some [features]... but it may be artificial to compare these with [Product B features], because of the magnitude of [features], and what's involved." [Interview C.001.CP]

Table 5.2.1 Differences between Project B and Project C

Characteristic	Project B	Project C
Size of support team	Support team of 50 people (separate from Project B).	Support team of 12 people (part of Project C)
Size of planned development team	approx. 38 people	approx. 3 people
Size of planned management team	approx. 6 people	approx. 3 people
Assignment of work between support team and development team	Developers are either support or development (but development may support in critical situations)	Developers 'own' components and both develop and support those components.
Role(s) of Project Leader	Project Leader	Project Leader, Design/Code Manager, Support Manager
Strategic value of product	Higher; long-term	Lower; mid- to short-term
Purpose of project	New functionality	Port to new platform
Type of product	Large, mission-critical, middleware legacy system	Large, middleware legacy system
Release sizes	36 KLOC	70 KLOC
Number of features/design changes	13 features (planned) 12 design changes (unplanned)	19 features (planned) and 11 features (unplanned)
Platforms	Mainframe	Workstation
Project status meetings	Yes	No, but design/code/test status meetings
Project duration (in weeks)	57 (planned and actual)	48 (planned) 59 (actual)
Product delivery week	52 (planned and actual)	48 (planned) 59 (actual)
Determination of project duration	Project end-date driven, due to market considerations	Project end-date driven, due to resource funding constraints

Table 5.2.2 Similarities across Project B and Project C

Characteristic	Comment
Business process	Both projects partially used the new business process
Organisation	Both projects were within the same laboratory
Composition of management team	Both projects used multi-functional project management teams, with representatives from each significant process area.
Project success	Both Project Leaders considered their projects to be successful.

5.3 The actual progress of Project B

Figure 5.3.1 presents information on the schedule, workload and capability of Project B, at the project-level, and shows how the actual progress of the project, with regards to these three constructs, differs from the planned progress. (See Appendix B2 for an explanation of the structure and notation of the figures in section 5.3 and 5.4.)

From the figure, it is clear that the plan, design/code and test phases (the three main phases of the project) all complete later than planned, with the design/code and test phases completing many weeks later than planned. In the case of the design/code phase, the phase lasts at least 50% longer than planned, although the comment on the completion of design change work in week 50 (see the 'Events' section of the figure) suggests that the design/code phase may persist in some form for almost the remainder of the project. Two design changes were also accepted, in week 37, after the actual completion of the design/code phase. This is discussed in more detail below.

In the case of the test phase, Figure 5.3.1 indicates that the test phase continues until week 58, six weeks after the product was actually delivered. This is because two features are being tested and delivered, at a later date from the rest of the product, via the World Wide Web. Note also that the design/code and test phases proceed concurrently for a number of weeks, once again differing from the planned, sequential progress of the project.

In addition to the differences between the planned and actual progress of phases, note the frequency of planned milestones for the project. With only two exceptions, the design complete milestone and the functional verification complete milestone, all milestones are planned to occur during approximately the last quarter of the project. Two of these milestones are project oriented (the system test and integration complete milestones) whereas the other two milestones are business oriented (the availability and announce checkpoints). Consequently, for long periods of the project there are no high-level checks of how the project is progressing. This may be because progress in the design phase is difficult to properly assess, and so even if there were milestones, these milestones would be ineffective. Abdel-Hamid ([2]) argues that reports of actual progress often simply reflect the planned progress because a more accurate assessment of actual progress is not possible.

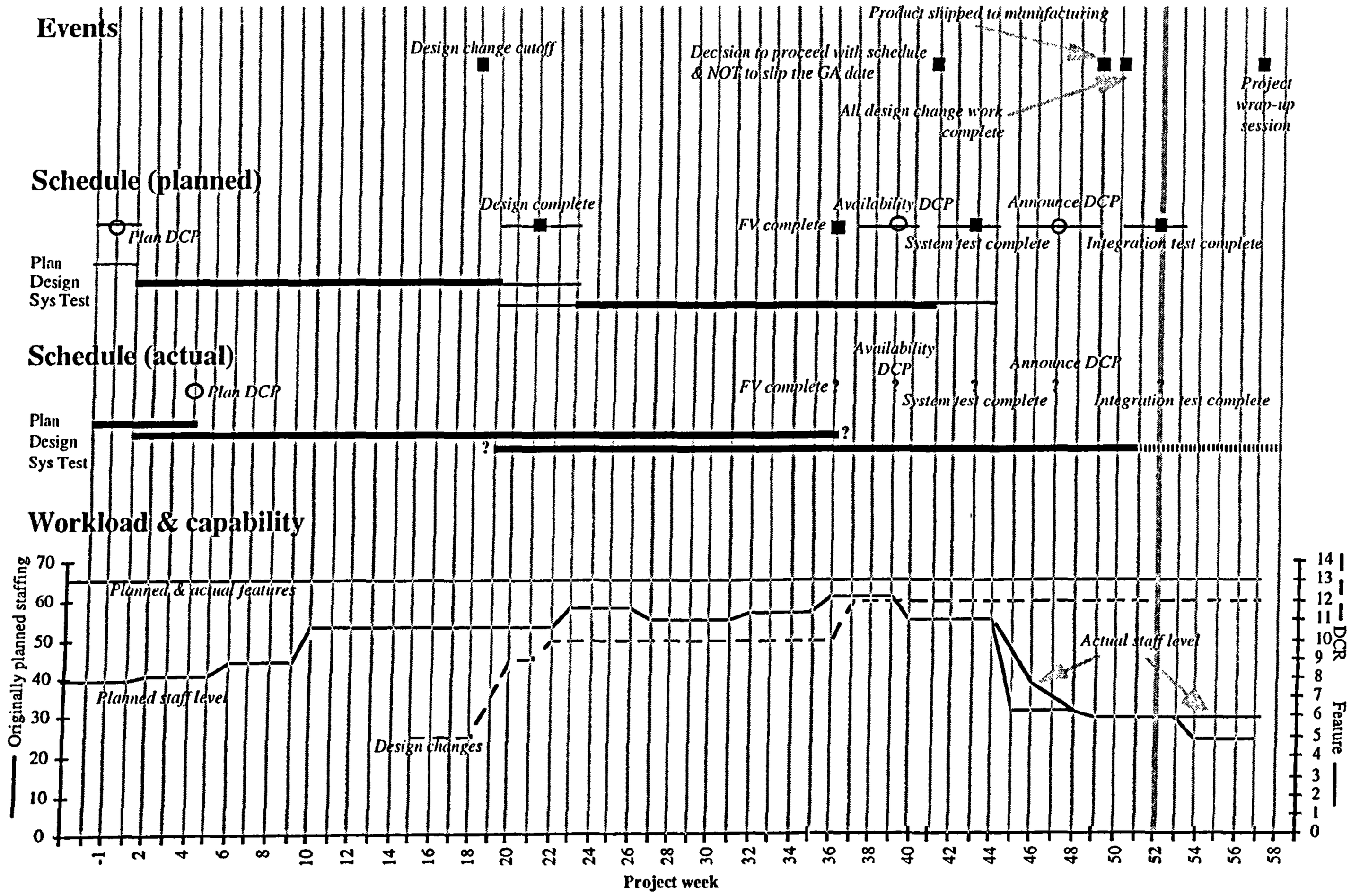


Figure 5.3.1 Planned and actual project schedule, project workload and project staffing for Project B

Workload is represented, in the figure, in terms of features and design changes. (These are measured using the scale on the right of the figure, with range zero to 14.) Broadly, both features and design changes are sets of market requirements of a piece of software which "... typically involve changes and additions to multiple [software] subsystems" ([121], p. 840). Whilst, in principle, features refer to new functionality and design changes refer to modifications to existing functionality, in practice there are no clear distinctions between a feature and a design change. (In terms of code size, a design change may be larger than a feature.) At the feedback workshop, the Project Leader provided more information on features and design changes:

1. Features are the work that is planned at the beginning of the project.
2. Changes in workload, once the project starts, are managed as design changes.
3. Some of the design changes accepted on to the project were actually features, and some of these are larger in size that the 13 features originally planned. (This indicates how features and design changes are not effective measures of process size or product size.)
4. There are two types of design changes:
   - design changes that add function.
   - design changes that remove function.

   Both types involve work i.e. they increase workload on the project. The first type increases the size of the product. The second type reduces the size of the product.
5. A very low number of design changes were expected for Project B (almost zero).

Figure 5.3.1 indicates an increase in workload on the project, with the number of design changes increasing from zero to 12. This is a near-100% increase in the combined *number* of design changes and features, although this does not necessarily imply a 100% increase in the actual amount of workload (because features and design changes are not reliable measures of process size and product size). Recall, from section 5.2, that the Project Leader has a policy of rejecting all design changes, although he concedes that there are some design changes that the project will have to do.

In addition to the increase in the number of design changes, note the timing of these increases. The intended last week for accepting design changes is week 18, close to the planned completion of the design/code phase, but at this point only five of the eventual 12 design changes are accepted. (The remaining seven are being considered by week 18.) Furthermore, note that some design changes are accepted after the actual completion of the design/code phase. As discussed earlier, this indicates that the design/code phase may actually progress for longer than represented in Figure 5.3.1. The increase in design changes after the actual completion of the design/code phase also suggests that the

project is in multiple phases at any one time (*cf.* [92]) and that the plan does not represent these multiple phases accurately.

Capability is represented, in the figure, in terms of weekly resource levels. (In the project documents for Project B, planned resource levels are recorded on a monthly basis, so these have been converted to weekly resource levels for Figure 5.3.1. The scale on the left of the figure, with range zero to 70, is used to measure weekly staff levels.) At the feedback workshop, the Project Leader annotated an earlier version of Figure 5.3.1 to indicate the actual weekly staff levels for his project. It is clear from Figure 5.3.1 that there are some slight increases in staff, but rather than adding people to the project (*cf.* Brook's Law), the Project Leader is able to delay the re-assignment of existing staff to their new project (Project B+1). This is similar to Perry *et al.*'s ([92]) observation that designers are reassigned to higher priority projects, in this instance remaining with a project rather than being assigned, as planned, to another project.

The planned ramp-up of staff does not exhibit the left-skewed bell-shape curve assumed by Raleigh/Putnam-based prediction models (e.g. [97]). This may be because, as noted earlier, Project B is costed as one of four projects. Also, Project B starts after Project B+1 starts, but completes before the completion of Project B+1. The lack of symmetry between the planned ramp-up and planned ramp-down suggests that designers and system testers are gradually being assigned to the project (over weeks 1 through 23), but are then abruptly unassigned from the project in week 45.

Overall, Figure 5.3.1 shows that, at the project-level, the actual progress of Project B is different to that planned. Figure 5.3.2 shows how the actual progress of two particular features (F02 and F03) also differ from their planned progress. These two features are presented because, of all the features in the project, the most information is available on these two. This is because these two features are discussed most in the project status meetings, which is probably because they are the most problematic features on the project.

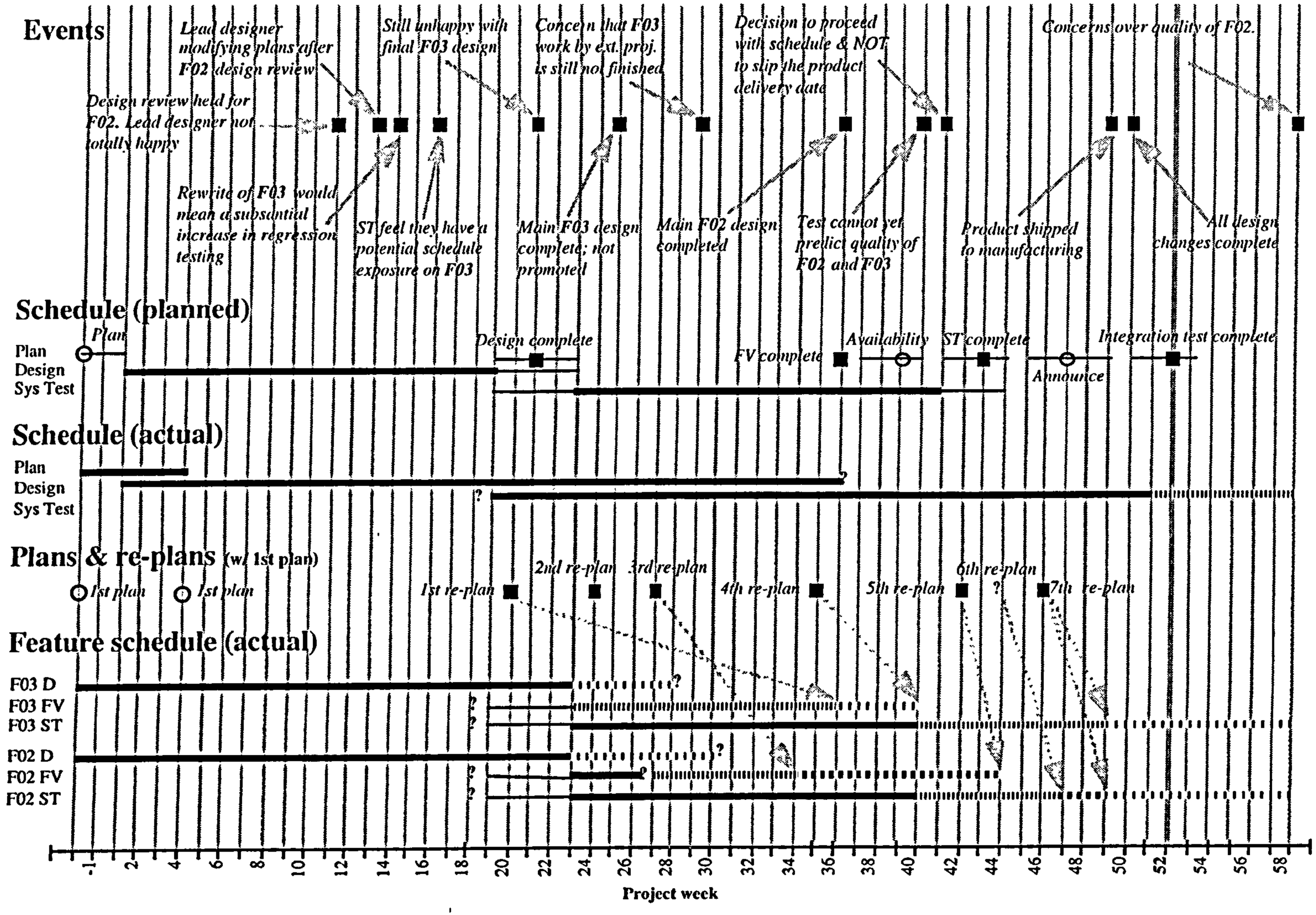


Figure 5.3.2 Actual feature schedule for two features of Project B

Figure 5.3.2 shows that of seven re-plans on Project B, six refer to re-scheduling the phases of features F02 and F03. (The other re-plan concerns the re-scheduling of feature F07, the software transaction logger.) Consistent with these re-plans, both features complete their respective design/code phases, functional verification phases and system test phases later than originally planned, and are the two features delivered via the World Wide Web. These two features most clearly impact the project-level schedule.

With regards to the frequency of the re-plans, 20 weeks of the project pass before the first re-plan occurs, but then six further re-plans occur during the next 26 weeks. This is, on average, every 4.3 weeks i.e. one plan per month. (The exact week when the sixth re-plan occurs is not known, but this does not affect the calculation of the average.)

The comments presented in the 'Events' section of the figure indicate that both features experience significant re-designs early in the project. In addition, designers of feature F03 are concerned that required work by an external project will not be finished in time. This indicates that an external project may contribute to the delay in completing the design of feature F03, because the designers are waiting on the delivery of code from that project.

The events, the number of re-plans and the progress of the features' phases all suggest either that the capability for these features is less than planned and/or the complexity, and hence workload, for these two features is greater than planned. The re-plans are an 'internal' adjustment (internal because they did not require senior management approval) to compensate for the differences in capability and/or workload.

Related to the re-plans are a series of events identified in this thesis as 'indicators of project activity'. These events suggest a number of tactics used by management to respond to discrepancies between the planned progress and the actual progress: tactics to reduce workload, tactics to increase capability, tactics to re-distribute capability, and tactics to iterate a process more frequently. As a result, these tactics suggest that the project's management recognise that in some way the project is becoming more urgent or more 'active'. The concept of project activity incorporates a psychological element in addition to a technical element. Tactics of management are discussed in section 5.5.

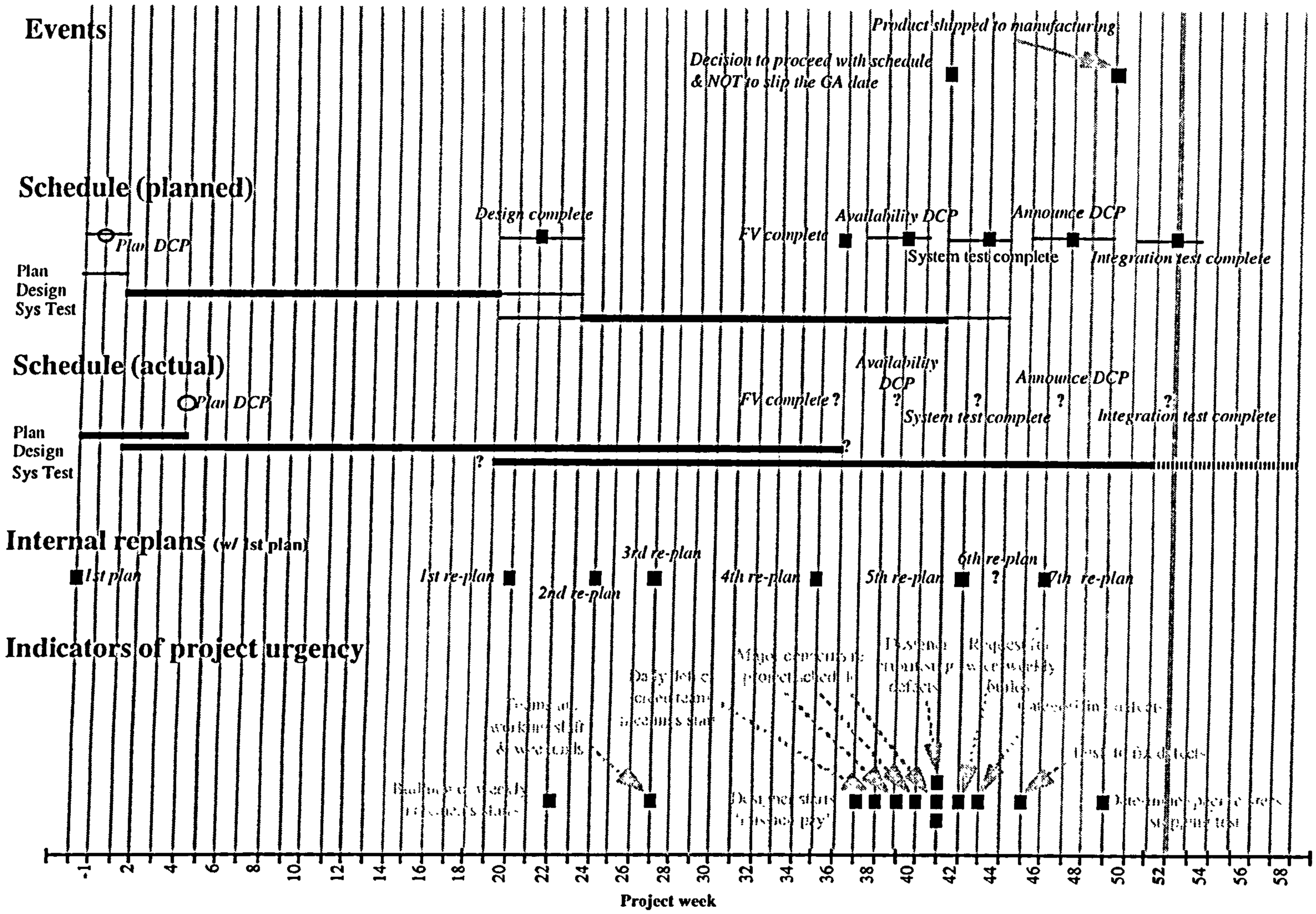


Figure 5.3.3 Internal re-plans and indicators of project activity for Project B

Figure 5.3.3 shows the indicators of project activity. Of the 13 indicators, only one seems to be planned (the building of weekly increments from week 22). Many of the other indicators are 'situated', in that their presence depends on how the project 'unfolds', and as a result, these indicators cannot be planned for (although they might be expected). Most of the indicators occur as the project approaches the planned completion of test (between weeks 37 and 43). At the 'centre' of these indicators (i.e. week 41), the project's management commit to the original product delivery date, rather than seeking to re-negotiate (with senior management) a revised product delivery date. At the feedback workshop, the Project Leader explained various reasons for committing to the original date:

> By this stage of the project, people have 'project blues' and just want to finish the project. Maintaining the current schedule helps to maintain current morale and prevents a further drop in morale (which would arise if the schedule was adjusted). This, in turn, maintains productivity. Furthermore, planning for a slip (i.e. adjusting the schedule) means that the project will slip. Its a self-fulfilling prophecy: work will expand to fill the allotted time. Furthermore, if the project actually slipped, it would only have slipped by about a month, and that was a gamble worth taking. The project could use 'other channels' (e.g. marketing) to manage the effects of the project slipping.
>
> [Feedback workshop B.FW.001; this is not a verbatim quote]

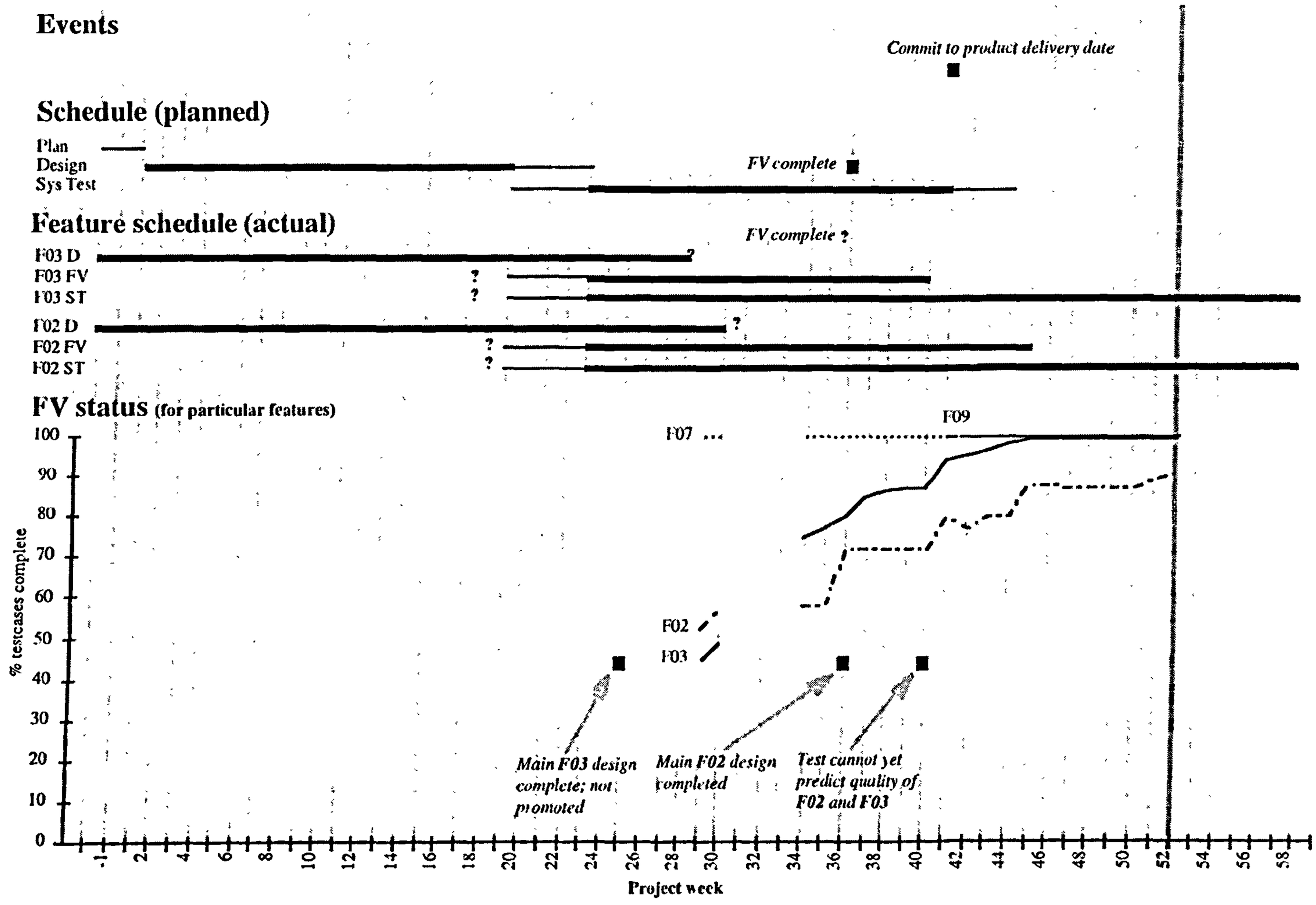


Figure 5.3.4 The status of functional verification testcases for four features in Project B

The point was made, toward the beginning of this section, that there is an increase in workload (specifically, an increase in design changes) for Project B. The late completion of the design/code and test phases are evidence that the actual capability of the project is not 'balanced' with the actual workload (so that the work cannot be completed within the original time-frame). The suggestion has already been made that management make efforts to increase capability and reduce workload (see the indicators of project activity) in addition to re-organising the planned completion of the phases (i.e. the internal re-plans). Figure 5.3.4 provides further evidence of the project management's efforts to reduce the workload. Of the four features shown in the figure, two features (F07 and F09) appear to have completed their functional verification testcases uneventfully. The functional verification of a third feature (F03) appears to be problematic (reflecting the need, in Figure 5.3.2, to re-plan the completion of the functional verification phase for that feature). The functional verification of the fourth feature (F02) is so problematic that the functional verification was not completed for this feature. Not completing the functional verification is a method for reducing the workload on the project. (Despite not completing all of the testcases, it is likely that the functional verification test team would have prioritised the testcases, ensuring that the more important testcases were completed.)

Closer inspection of the functional verification testcase status for feature F02 suggests a cyclic pattern of increases and 'plateaus' i.e. an increase at week 36 followed by a plateau of four weeks; an increase at week 41, followed by a plateau for three weeks; an increase at week 45, followed by a plateau for five weeks. This behaviour might reflect the dependency of the functional verification process areas on the design/code process area, where design/code release code to functional verification every few weeks and functional verification test the new code. At a more detailed level again, this behaviour suggests that the functional verification process area (and, indeed, the design/code process area) is constantly re-distributing its effort, attending to other features and periodically returning to attend to feature F02. This re-distribution of effort would be another example of the project attempting to increase its capability by allocating effort more effectively.

Another example of re-distributing effort is the management of defects. Figure 5.3.5 presents information on defect status for Project B. Defects pass through several states. When a problem is identified in the software, a defect is 'opened'. The problem is then investigated to confirm that the problem is a defect, and if it is a defect it is then 'accepted' as a defect. The defect is then assigned to a defect-fixer, who will then seek to provide a fix to the defect. When a fix is developed, the defect is re-assigned to the 'answered' state. 'Answered' defects are then selected for re-testing, where they are re-assigned to 'test fix'. With a successful fix, a defect leaves the defect system. An

unsuccessful defect is then returned to an earlier state in the process (depending on the nature of the defect). The relatively high number of defects at the beginning of the project is due to 'residual' defects from previous releases and from customers. Units for the y-axis in Figure 5.3.5 are not shown for reasons of confidentiality.

The statement at week 38, regarding the daily defect meetings, indicates that the status of defects is becoming an increasing concern. (Note that the statement at week 38 was also used as an indicator of project activity.) At week 38, effort on the part of the Defect Screen Team, and in the form of daily meetings, is increased to attend to these concerns. In the weeks immediately following the commencement of daily defect meetings, there is a dramatic decline in the number of 'opened' defects and an increase in the number of 'accepted' defects. This reflects the fact that the Defect Screen Team have 'promoted' defects and assigned them to a designer.

In the same vein, the statement at week 45 suggests that the project once again re-prioritises its work, this time focusing on *fixing* defects for the manufacturing build. With this re-prioritisation of work, it appears that a reasonably large number of defects are 'promoted' through to the next stage of the defect process i.e. a number of opened defects are 'promoted' to accepted defects; a number of accepted defects are 'promoted' to answered defects; and a number of answered defects are 'promoted' through to defects with fixes awaiting re-testing. The 'blip' in the number of defects with fixes awaiting re-testing ('Test fix') at week 46 reflect first an increase in the number of fixes awaiting re-testing (caused by the 'promotion' of defects) and then a decrease as these fixes have been tested and either accepted as fixes, or returned to the accepted stage (note the subtle increase in the number of accepted and answered defects between weeks 46 and 48). The number of 'opened' defects increases, perhaps because attention is directed at fixing defects and potential defects (i.e. problems) are temporarily neglected.

Overall, there appears to be two broad stages to handling defects, these stages being triggered by the number of defects in particular states and the approach of the manufacturing phase. In the first stage, project management focus on allocating defects so as to reduce the number of opened defects and to accelerate the fixing of defects. In the second stage, development focus on fixing their allocated defects so as to reduce the number of outstanding defects.

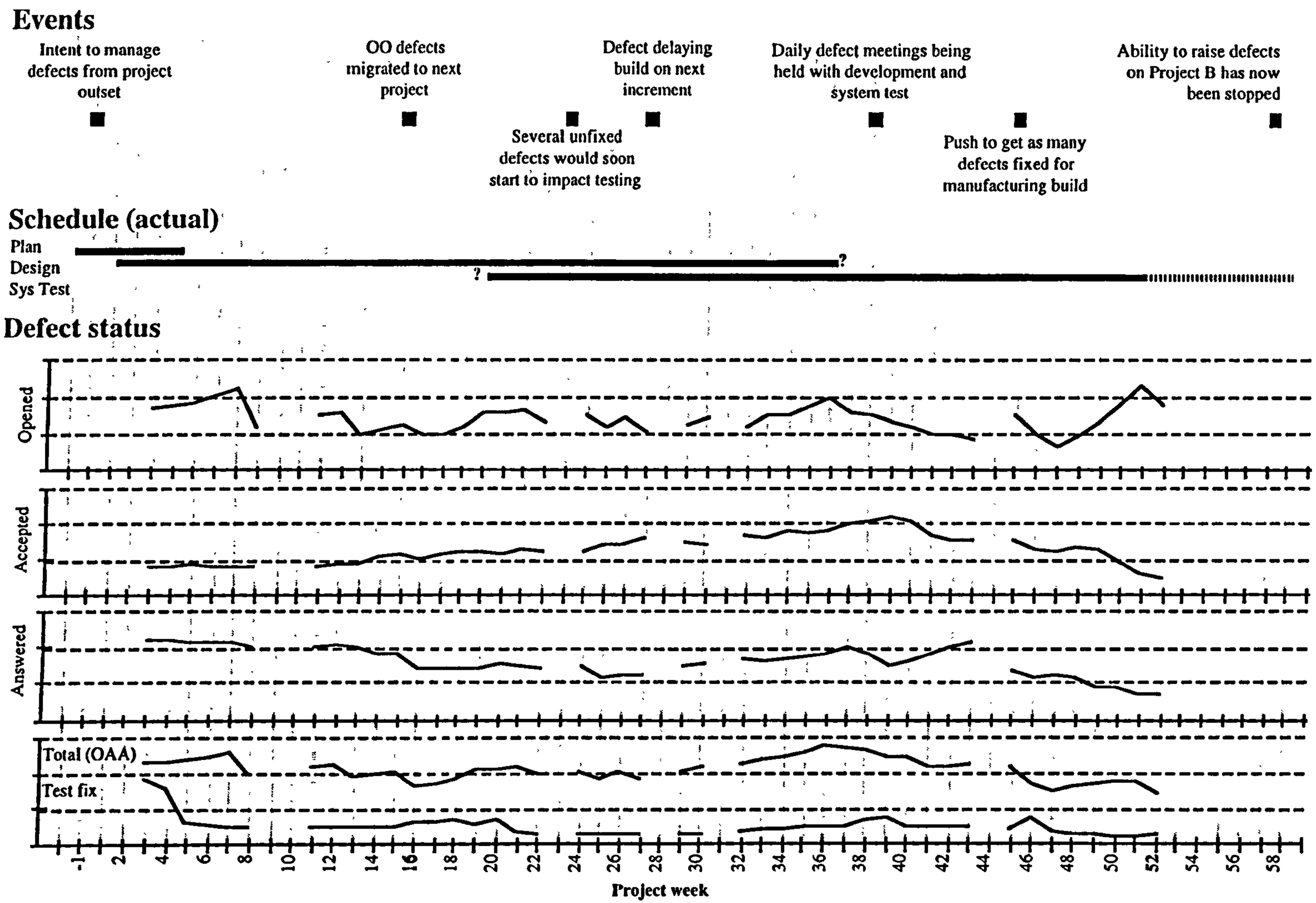


Figure 5.3.5 Status of defects in Project B

5.4 The actual progress of Project C

Figure 5.4.1 presents information on the schedule, workload and capability of Project C (at the project-level) and shows how the actual progress of the project, with regards to these three constructs, differs from the planned progress. (See Appendix B2 for an explanation of the structure and notation of the figures.) Table 5.4.1 presents information compiled from the feedback workshops for Project C. Most of the events in Table 5.4.1 refer to events in the 'Events' section of Figure 5.4.1, and document the effects these events had on the actual workload, capability and schedule of the project. Included in the table is a 'template' of the questions asked and the permitted answers to these questions. Comments to clarify responses were allowed and these are reproduced in the table. The focus of the workload-related questions is not on whether events increase or decrease the quality, functionality and performance of the product but whether the events increase or decrease the workload *required to deliver* quality, functionality and performance.

From the figure, it is clear that the project does not complete when originally planned and actually completes 11 weeks later. This is a slippage of about 20%, which McKeen ([76]) found to be quite typical for software development projects. It is also clear that the plan, design/code and test phases (the three main phases of the project) all complete later than planned, with the design/code and test phases completing many weeks later than planned. In the case of the design/code phase, the phase lasts approximately 80% longer than planned. In the case of the test phase, the phase continues until approximately week 55, seven weeks after the product is originally planned to be delivered, and also approximately 80% longer than planned. The figure also shows that the manufacturing phase actually compresses, from seven weeks down to four. Note also that the phases are originally planned to progress in a sequential manner, but they actually progress concurrently. Despite the extension to the project duration, Project C, like Project B, still delivers some features via the World Wide Web. Unlike Project B, however, these features are delivered at the same time that the product completes, rather than several weeks later.

Table 5.4.1 shows that, almost without exception, the major events of the project affect the project's schedule. The Project Leader explained (although this not shown in the table) that individually these events could be contained within the original plan, but as a group they could not be.

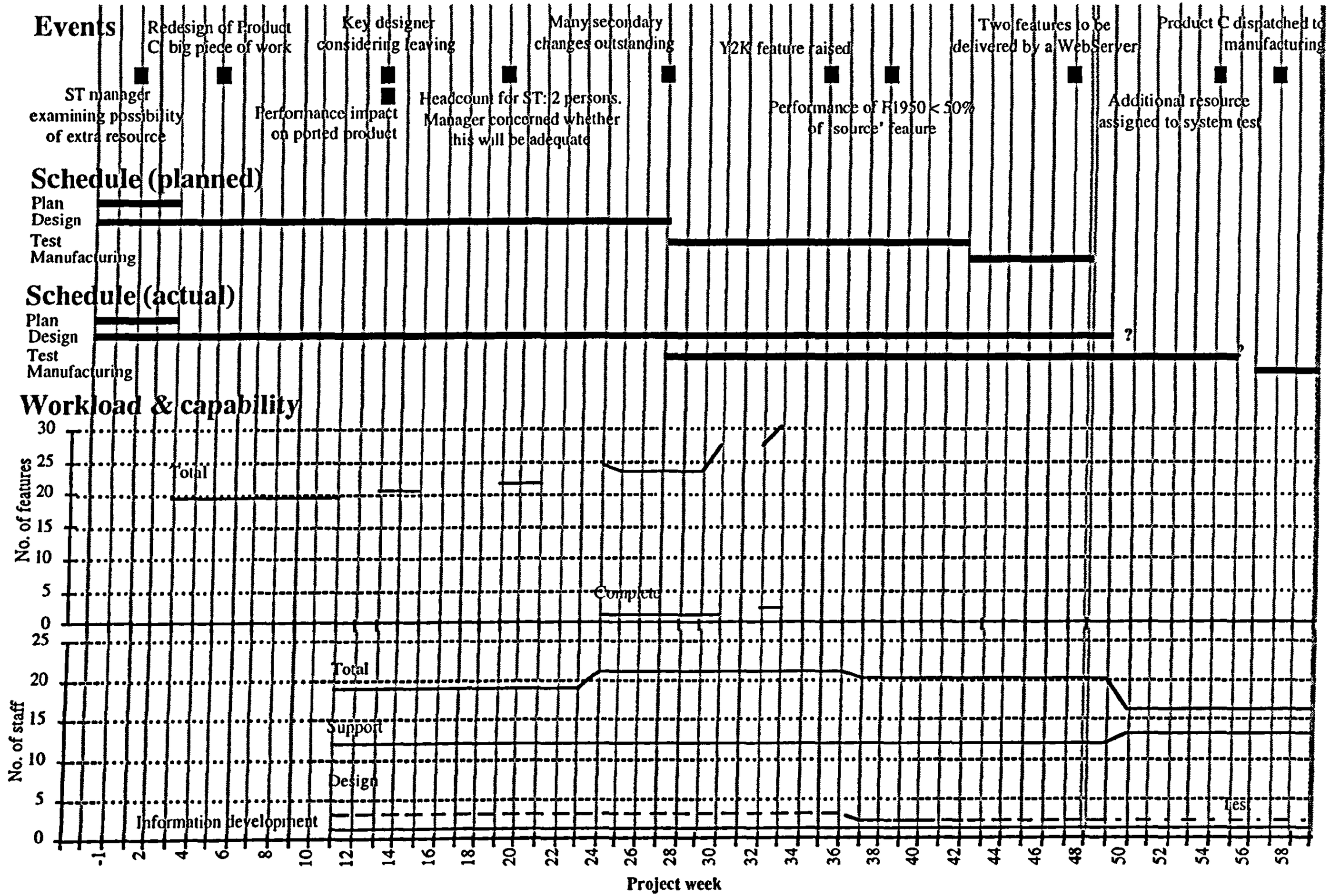


Figure 5.4.1 Project-level schedule, workload and capability for Project C

Table 5.4.1 The effect of certain events on development workload, capability and schedule for Project C

Event	Development workload			Development capability	Development schedule	Comments
	Quality	Functionality	Performance			
Major re-design	Increase	Major increase	No change	No change	Increase	Additional requirements as to how the product runs on the new operating system
Key designer stating their intention to leave	No change	No change	No change	Decrease	Increase or No change	Impact on team motivation / morale
Performance 'problems' emerging System Test	No change	No change	No change	No change	No change	
Headcount	Increase	No change	No change	Decrease	Increase	
Skills	Increase	No change	No change	Decrease	Increase	
Vacation during System Test phase	No change	No change	No change	Decrease	Increase	
New year-2000 requirements	Increase	Increase	No change	No change	Increase	
Project Leader: one person doing three jobs	Increase	Increase	Increase	Decrease	Increase	
Many secondary changes	Increase	Increase	No change	No change	Increase	Requirements 'creep'
Support work	No change	No change	No change	Major decrease	Increase	

Question template: "Did the event increase/decrease...

the development and test workload required to deliver a quality product
the development and test workload required to deliver functionality
the development and test workload required to deliver performance
the development and test capability
the development and test schedule"

Responses were restricted to:

Major increase
Increase
No change
Decrease
Major decrease
Don't know

Workload is represented, in Figure 5.4.1, in terms of features (see section 5.3 for a definition of features; design changes were not used in Project C). It is clear from the figure than the actual workload increases from 19 features to 30. (Gaps in the recording of features are due to status meetings typically occurring fortnightly, and some status meetings not recording the progress of the features. From week 33 until the end of the project the progress of features is not recorded.) This is an increase of almost 50% in the *number* of features but, as with Project B, this does not imply a near-50% increase in workload. For example, discussion with the Project Leader (in interview C.008.CP) indicates that some initial features are subsequently separated into two features in order to help manage the workload.

The figure also indicates that it is not until week 24, half way through the original duration of the project, that the first feature is recorded as being completed; and by week 33, approximately three-quarters of the way through the original duration of the project, only two features are recorded as being completed. Three possible reasons for the apparent late completion of features have been identified:

1. Work is not completed at a uniform rate. A technical planner explained:

   "With applications it might be easier to develop a function incrementally. With middleware you might need to develop the whole thing before anything tangible results." [C.004.CR]

2. The Project Leader explained (in interview C.008.CP) that the feature tracking process had not been as rigorous as it might, and that people were more concerned with developing the feature than ensuring it was tracked properly. (Project planning and control appears to be traded against production capability.)

3. There was confusion as to what exactly the term 'complete' meant: whether the design and code for a feature was complete or whether the feature was designed, coded and sufficiently tested.

In addition to the clear increase in the number of features, it is also clear that there is a major re-design of the product in week 5, shortly after the plan was accepted. Table 5.4.1 indicates that this increases the workload required to deliver quality, and causes a major increase in the workload required to deliver the functionality. The re-design also increases the duration of the project.

Capability is represented, in Figure 5.4.1, in terms of weekly staff levels. (Staff levels were planned on a quarterly basis and these have been converted to weekly values for the figure. This explains the 'plateaus' in staff levels.) As explained in Section 5.2, most of the project resource is committed to supporting the previous releases (i.e. support), rather than developing a new release. This is reflected in the high number of support staff (12) and the low number of development staff (three). There are only two members of staff assigned to system test, and they are assigned for only the second half of the project (from week 37). The total number of staff on the project does not vary for the duration of the project. The *distribution* of the staff, however, does vary considerably. Evidence (from interview C.009.CP) indicates that approximately 50% of the total resource (approximately 8.5 person-years effort) is actually involved in development. The re-allocation of support personnel to new development is an example of a management tactic for dealing with project workload. As a related example, the Project Leader explained:

> "We are constantly juggling work assignments to even the workload." [C.007.CP]

Together, these are examples from Project C of behaviour first identified in Project B i.e. re-allocating effort to make that effort more effective.

Table 5.4.1 also indicates that the Project Leader believes that the support workload has a major impact on the development capability. As already explained, although three people are formally funded for new development, many more than three people are actually involved in new development. Support work prevented these additional people progressing with their new development work. (There are numerous references, within the status meeting minutes, to support work interrupting new development work.)

Besides the support work, Table 5.4.1 indicates that actual capability is lower than planned because the key designer intends to leave the project, because there are insufficient numbers of skilled system testers, and because the Project Leader has three roles to fulfil. With regards the intention of the key designer to leave, the Project Leader reflects:

> "The biggest problem has been the situation with [the key designer]. However, this is not just a resource problem, but a skilled resource problem. One can't just replace [that designer] with someone else, because they have a lot of skills and experience." [Interview C.007.CP]

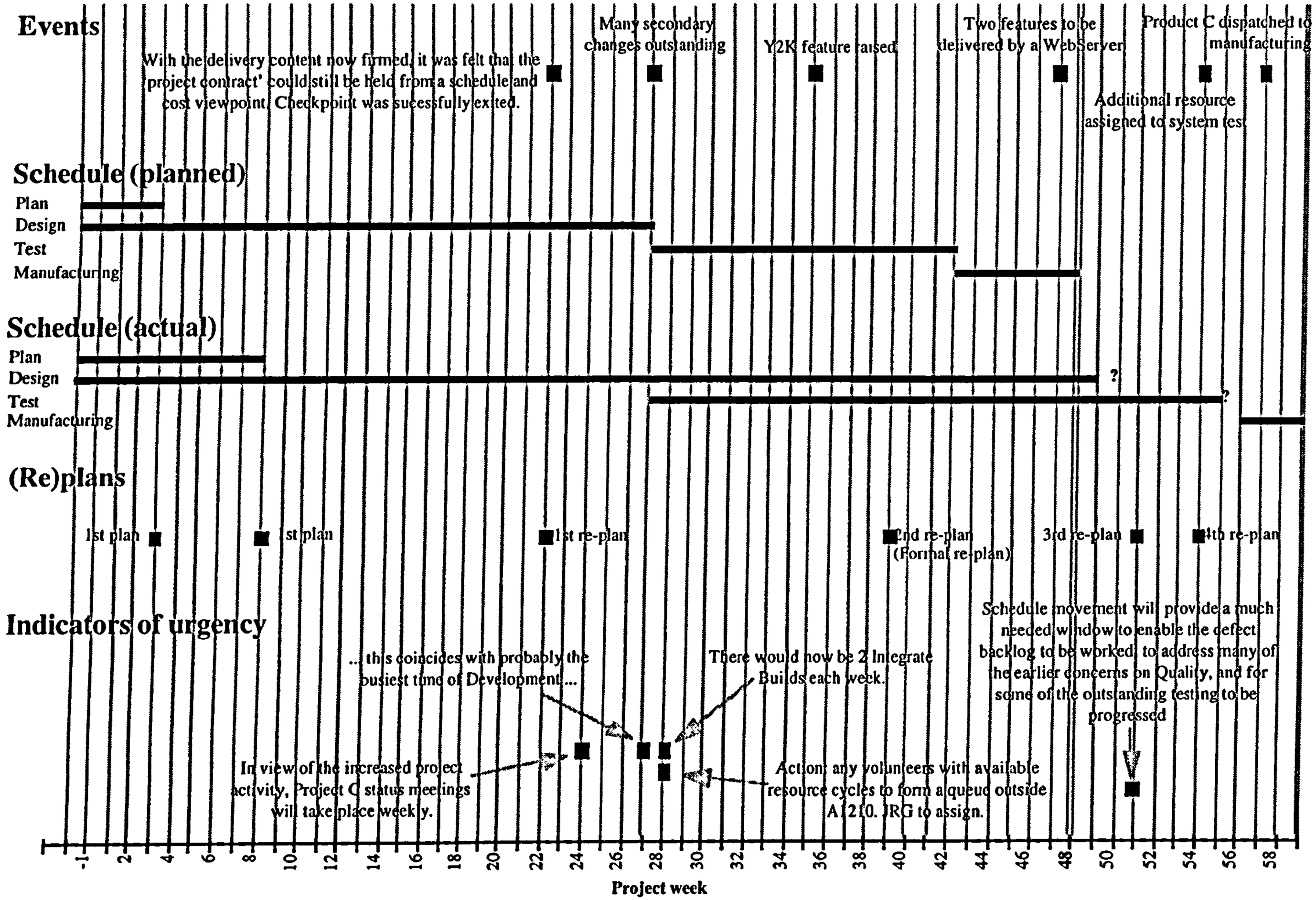


Figure 5.4.2 Re-plans and indicators of project activity for Project C

With regards to the insufficient number of skilled system testers, these are identified as a risk and discussed in section 5.2. Figure 5.4.1 shows that, in week 54, resource is re-assigned to system testing. This is an another example of behaviour first identified in Project B i.e. re-allocating effort to make that effort more effective. Section 5.2 also considers the problems the Project Leader experiences in fulfilling his various roles.

Figure 5.4.2 shows the re-plans and the indicators of activity for Project C. Unlike Project B, where all the re-plans are internal re-plans, Project C has one external re-plan in which the plan is formally re-negotiated with senior management. This formal re-plan occurs in week 39 and results in an extension to the completion of the project, from week 48 to week 59. (From the perspective of the model of schedule behaviour, the remaining-duration has increased, in week 39, by 11 weeks.) The formal re-plan is caused by the introduction of new Year-2000 requirements earlier in the project, resulting in the introduction of a new feature in week 35. The two plans toward the beginning of the project, both labelled '1st Plan', are due to the fact that a second Plan Decision Checkpoint was required (in week 8); the second plan addressing revenue issues rather than schedule issues.

In the first re-plan (week 22), the project team believe that the schedule can be held (see the corresponding event for that week). Two weeks later, the frequency of the status meetings change from fortnightly to weekly, suggesting an increase in project activity. One possible explanation is that the project team believe that while the schedule is still attainable they will need to be more 'focused' in their work i.e. need to increase capability through working harder and smarter, and/or through allocating effort more effectively.

The statement regarding volunteers with available resource cycles (week 28) is ironic because, by this stage of the project, all team members are fully assigned to work.

The increase in the number of integrate builds (week 28) might be because the test phase is starting, or because the design phase and test phase are proceeding concurrently (where it is important to quickly transfer completed design work over to test). Note that, like Project B, the original plan consists of sequential phases rather than concurrent phases.

The comment on schedule movement (in week 51) refers to a schedule movement in the start of the manufacturing phase. The test phase is still incomplete (with outstanding defects and test cases), and the manufacturing phase is compressed to provide more time for test. The resource re-assigned to test (in week 54), from elsewhere in the project, appears to be a response to the outstanding test issues. Recall from section 5.2 that the

Project Leader, The Project Assistant and the Test Manager are all concerned with the test process area.

Figure 5.4.3 provides detailed information on the re-plans and their effect on the phases of the project. The first re-plan appears as a response to the delay in completing the design phase. The schedule is adjusted in an attempt to cause minimal disruption to the system test and manufacturing phases. In the period between weeks 27 and 31, the design and acceptance test phases are planned to proceed concurrently. From week 32, at which time the design phase is expected to complete, the sequential order of phases is planned to return. This first re-plan consists of re-organising (another management tactic) the project work into two groups. The first group of work consists of the OS/2 and DOS work. The fact that test (acceptance test and system test) will address this work first suggests that this work is progressing well through the design phase (or at least is believed to be progressing well), and that it will progress well through the test phases. 'Pushing' the work relating to the new product, the second group of work, back in the test schedule suggests that this work is more troublesome in the design phase and may be more troublesome in the test phases. One conjecture is that the design problems with the new product are caused by the need for a re-design, identified in week 5 (see Figure 5.4.1).

As already explained for Figure 5.4.2, the second re-plan (the formal re-plan) extends the project duration by 11 weeks. Around the time of this re-plan, the Project Leader believes that the project can be completed by week 48, as originally planned, if the new Year-2000 requirements were not introduced (this was stated in interview C.009.CP). During the feedback workshops, the Project Leader reflects that the introduction of year-2000 requirements (imposed by the organisation on the project) was a fortuitous event for the project because it provided much needed additional schedule.

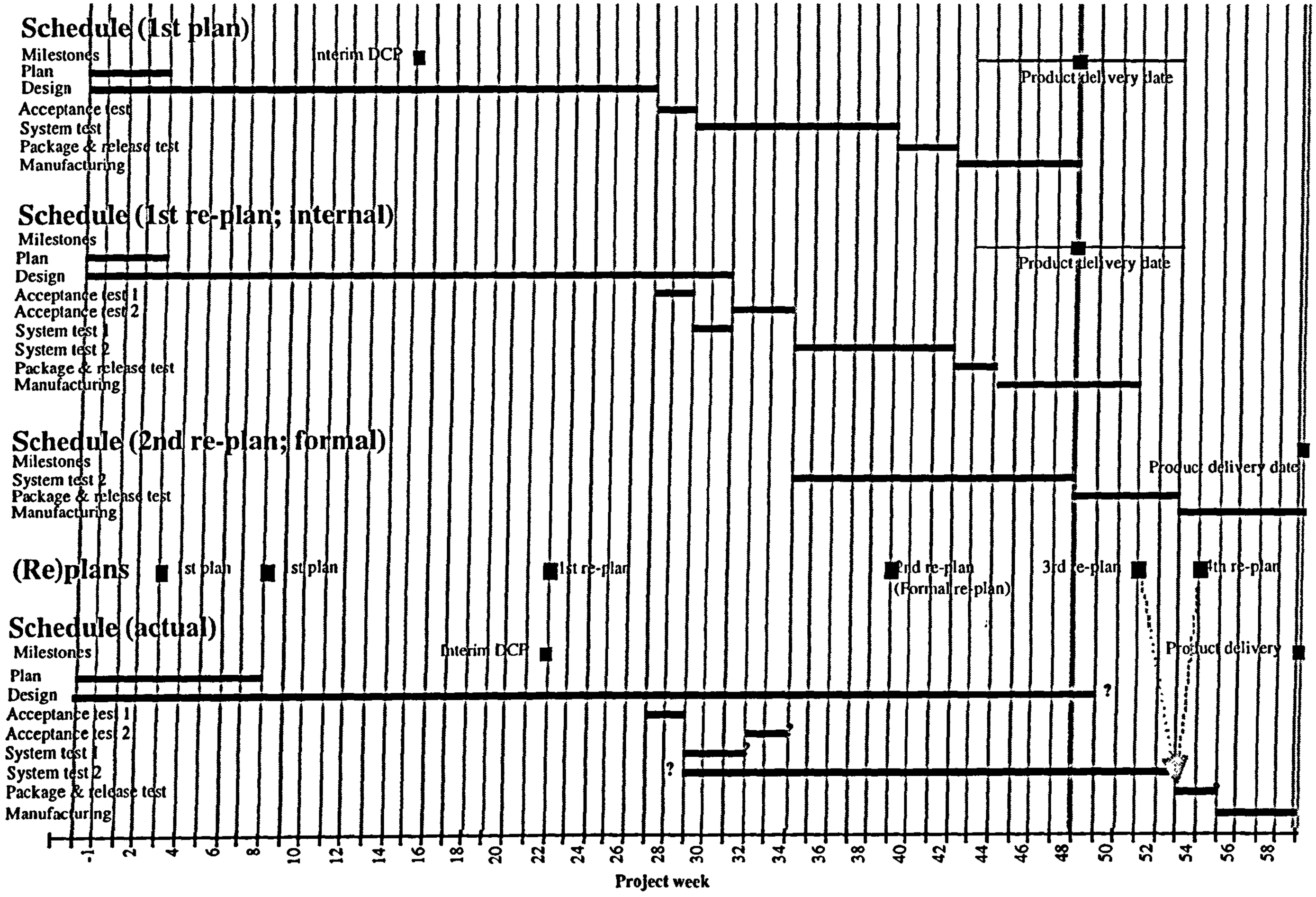


Figure 5.4.3 Re-plans and phase-level schedules for Project C

5.5 Tactics to manage the projects

The discussion and explanations of the actual progress of Projects B and C indicate that management use a number of tactics to respond to problems in their projects.

Table 5.5.1 A summary of some tactics used by the projects' managements

Tactic	Empirical examples
Re-prioritisation of work	"The severity of this [defect] will be raised from Severity 3 to Severity 2 as it is holding up further test measurements."
Re-distribution of work (re-allocation of effort)	"[BW]... reported that the [feature F02] approval task was currently being split amongst the team."
Re-definition of process	"Some testers are rearranging book approval dates because of holiday commitments - being done with the agreement of the writer concerned."
Separation of a phase into multiple sub-phases	"... [BR] has suggested a staged delivery plan for the new ... system that he is constructing."
Creation of more time/effort on the project	"[Feature F03]... [BB] to start work on [Project B+1] although testing F03 [code units and defects] will be his top priority."

An exhaustive list of the tactics used by the two projects' managements is difficult to identify and is beyond the scope of this thesis. Indeed, Sommerville and Rodden ([117]) argue that these kinds of behaviour are so diverse that it is not easy to represent them all. They write:

> "In fact, for a variety of reasons, people adapt these procedures to local circumstances and resource and this adaption (sic) is dynamic and responsive to change. Circumvention of the rules is the norm rather than the exception and the different kinds of circumvention are so diverse that they cannot readily be articulated." ([117], p. 55)

Table 5.5.1 provides some example tactics, together with examples of their use from the minutes of status meetings for Project B. The examples provided in Table 5.5.1 complement the tactics identified in sections 5.3 and 5.4.

5.6 Summary

As explained in the introduction to this chapter, the model of software project schedule behaviour has been used, from three different perspectives, to describe and explain the behaviour of Projects B and C. One interpretation is that the socio-technical contexts

provide an initial 'state' for workload, capability and duration, as well as a constraining framework within which these three constructs 'unfold' over the course of the project, and within which the projects' managements can use tactics to manipulate workload, capability and duration.

The socio-technical contexts of the two projects

Although a number of similarities exist between the two projects (see Table 5.2.2), with regards to their socio-technical context, it is clear that there are a considerable number of differences (see Table 5.2.1). This suggests that the two projects are constrained in different ways. Product B is, and has been, highly valued by the organisation. This means that the product area tends to get higher levels of capability, in terms of more resource and more highly skilled resource. Examples are: -

- Separate support and development teams.
- Considerably larger support and development teams.
- A larger management team.
- A considerably more experienced Project Leader.
- A Project Leader focused on only one role (that of project leadership).

Because of the legacy and value of the product, however, it has a number of dependencies with other products developed and maintained by the organisation. In general, the product also finds itself at the centre of political and strategic manoeuvrings, either because the product is a 'centre-piece' to the particular strategy or because it is a potential threat to that strategy. ('Political and strategic manoeuvrings' is not meant to suggest malicious behaviour; rather people pursuing what they strongly and honestly believe to be the best direction for a product and/or an organisation.) This means that the capability of the project is also constrained, and that the workload might be higher for these kinds of project (either through the number of software functions to provide, through the complexity of those functions, or the communication and coordination of technical dependencies between projects).

Product C, by contrast, is less valued by the organisation. (Since the completion of Project C, the product area is now managed under a different division of the organisation and as a consequence is much more valued.) This means that the product area tends to get lower capability, in terms of less resource and less skilled resource. In addition, there are occasions where skilled resource is transferred to other, more important product areas. Also, as developers develop their skills and experience so they want to move out of the

area into product areas with new technology. Examples of the impact of the lower value of the product area are:

- A funding constraint which leads to a resource constraint of 19 people for the year.
- The desire of the key designer to move out of the product area.
- The fact that the Project Leader has less experience of managing an entire project, and is required to fulfil three roles (i.e. Project Leader, Development Manager and Support Manager).
- The resource and skills levels for testing (recognised as a risk by a number of people in the project).
- A combined team for new development and support.
- The project end date driving the planning process.

The lower value of the product does, however, mean that there tends to be less dependencies on other projects, and the product area is more 'insulated' from the organisation's political and strategic manoeuvrings.

The actual progress of the two projects

Both projects are considered a success by their respective Project Leaders. For Project B, although the product is delivered when originally planned, two important features of the product are not delivered with the product and are delivered later via the World Wide Web. The eventual quality of one of those features is lower than desired and intended. For Project C, the project is considered successful despite the fact that the product delivery date is changed, so that the product is delivered 11 weeks later than originally planned. For both projects, there appear to be 'crunch moments' when the project managers realise that their schedules are at risk. For Project B, one of these moments occurs around week 41. For Project C, one of these moments occurs around week 39 (and resulting in the formal re-plan). The similar *timing* of these crunch moments for both projects is also an interesting observation.

For both projects, the design/code phases complete several weeks later than planned. The test phases start when planned, but complete several weeks later than planned. The design/code and test phases proceed concurrently for some time. There are also indications of an increase in project activity on both projects.

Both projects experience an increase in workload, in terms of features and/or design changes. Both projects also experience significant design problems during the design phase. For Project B, this is with the two features that were eventually delivered

separately from the product. For Project C, a customer prototype identified fundamental problems in the design of the product.

The tactics of management for the two projects

Both projects appear to adopt similar tactics in responding to poor progress and the increase in workload. Both projects take actions to increase their capability. This is partly through people working for longer on the project than planned, partly because people work much longer hours than contracted (thus disguising the actual amount of effort expended in the projects), and partly because effort is re-allocated so that it might be used more effectively.

Both projects also take actions to reduce their workload. This is through such tactics as re-prioritising and re-defining workload, and not completing lower priority work (for example, not fixing lower severity defects and not completing testcases).

Chapter 6 Waiting

6.1 Introduction

This chapter presents and discusses the analysis of the waiting evidence for Projects B and C. The waiting evidence is used in three ways. First, to provide insights into the nature of process areas within a project. Second, to provide evidence relating to the model of capability (recall from chapter four that waiting points to a preceding process difficulty elsewhere in the project and a succeeding 'threat' to capability). Third, to provide evidence to test Bradac *et al.*'s conjecture that waiting is more prevalent during the end of the project than during the middle of the project.

The specific research questions investigated in this chapter are:

- What is the frequency of references to waiting?
- What is the prevalence of waiting over the duration of the project? This is a test of Bradac *et al.*'s conjecture (See chapter two for more information on Bradac *et al.*'s work.)
- What are the different types of waiting, and what are their frequencies? This is also a replication of part of Bradac *et al.*'s study.
- What is the breakdown of 'source' and 'dependent' process areas.
- What are the relationships between the 'source' and 'dependent' process areas?
- What is the breakdown of the types of waiting against process areas of the project?

Details on the methods used to collect, organise and analyse the waiting evidence are provided in chapter three.

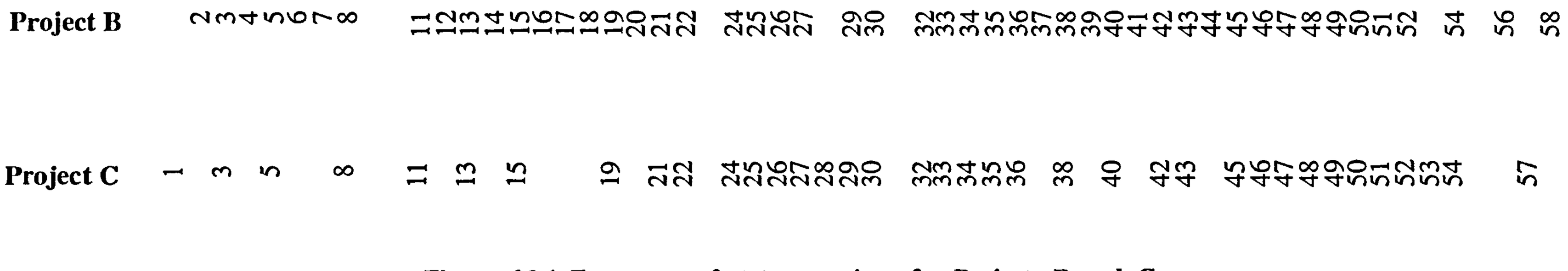


Figure 6.2.1 Frequency of status meetings for Projects B and C

6.2 A description of the evidence

As explained in chapter three, the waiting evidence is taken from the project status meeting minutes for Project B and the design/code/test status meeting minutes for Project C. Figure 6.2.1 plots the frequency of status meetings for the two projects. Where a meeting occurred, the week in which the meeting occurred is included in the figure. The figure indicates that both projects held frequent meetings, although Project B held meetings more frequently (typically weekly) than Project C (where meetings were initially held fortnightly, with a subsequent increase to weekly meetings).

The descriptions provided in Figure 6.2.1 are relevant to the analysis of waiting in this chapter, and to the analysis of the progress of work in chapter seven and the analysis of outstanding work in chapter eight.

6.3 The frequency and prevalence of waiting

The frequency of waiting

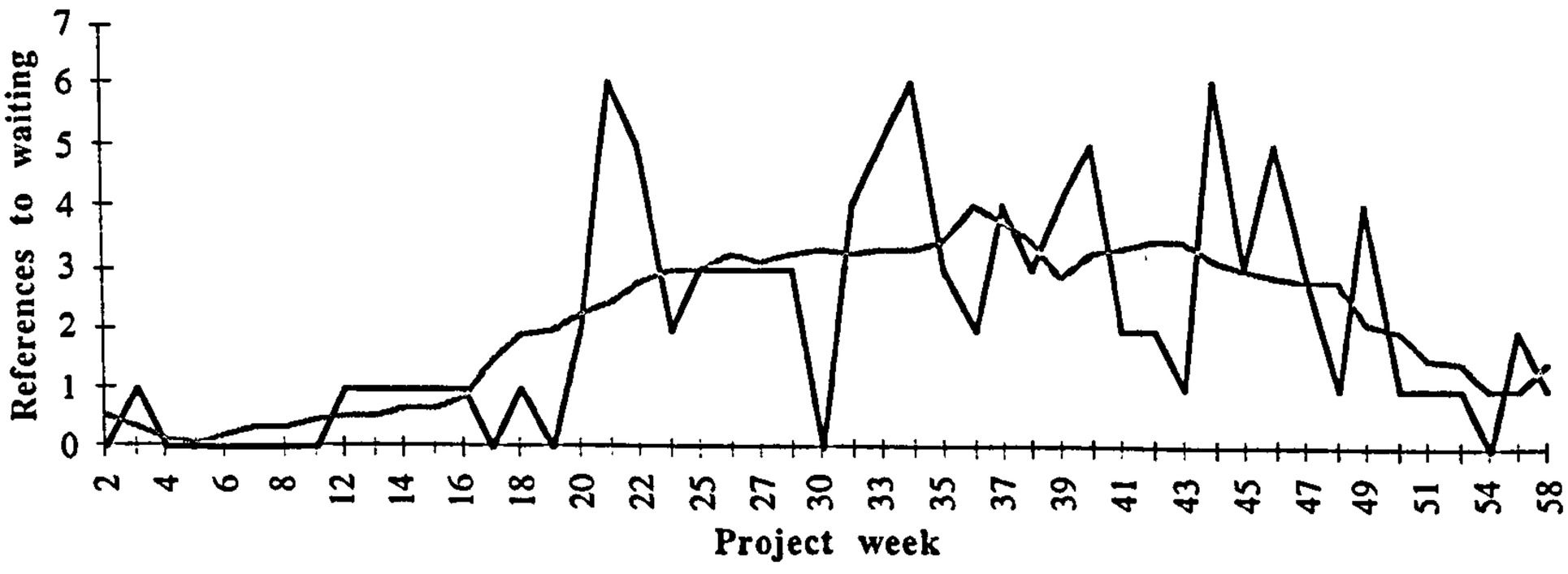


Figure 6.3.1 Frequency of references to waiting for Project B

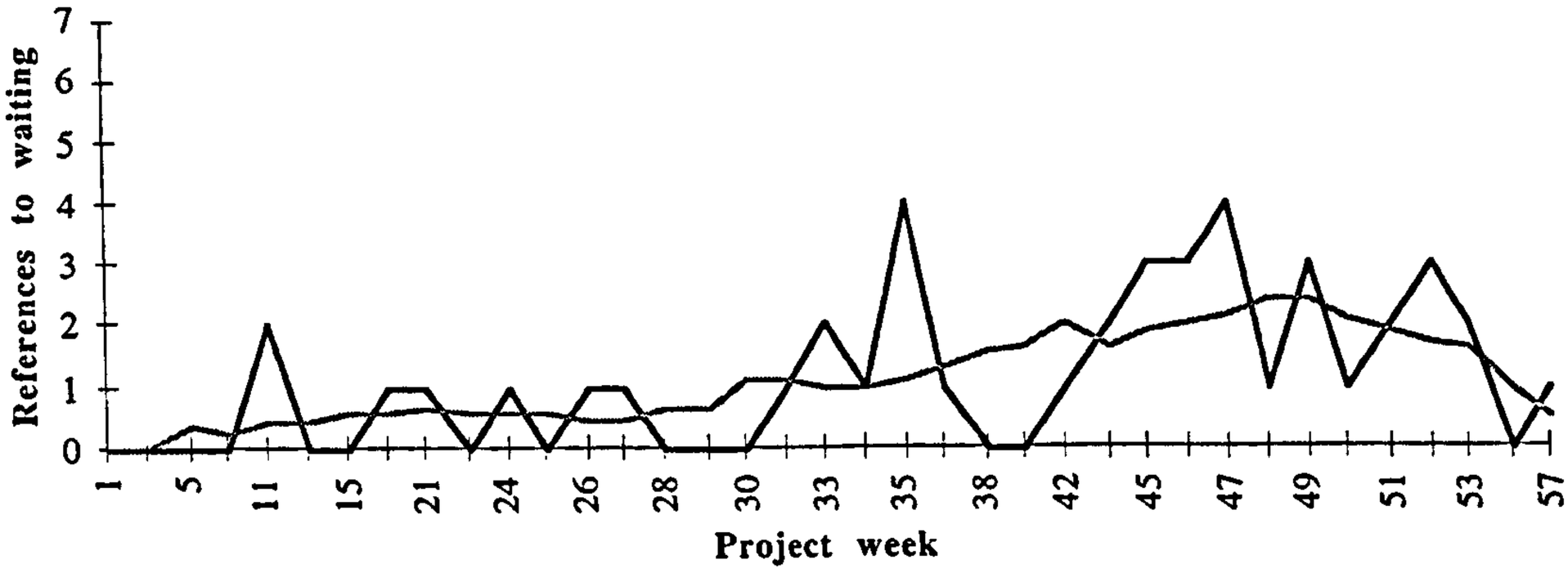


Figure 6.3.2 Frequency of references to waiting for Project C

Table 6.3.1 Summary statistics for the frequency of waiting

Project	Weeks	Mean	Median	Mode	Min	Max.	Range	Total
B	49	2.1	1	0	0	6	6	103
C	37	1.1	1	0	0	4	4	42

Figures 6.3.1 and 6.3.2 present lineplots of the frequency of references to waiting for Projects B and C respectively. The lineplots include a smoother, calculated as a moving average over a range of nine data points[3]. (Precise values are presented in Appendix B4.) Only the weeks for which a status meeting occurred are included in the lineplots. Table 6.3.1 provides summary statistics to support the two figures.

The table and figures show that the maximum number of references per week is greater for Project B than Project C (six compared to four), and that overall there are more references to waiting in Project B (103 compared to 42). This is partly explained by the greater number of status meetings for Project B (49 compared to 37). The two projects are identical in two of their central tendencies (median and mode) and their distributions (range), but the smoothers indicate a more noticeable increase in the frequency of waiting for Project B (around week 17).

For Project B, the increase in waiting begins in week 20 and persists through to week 50, although there are noticeable fluctuations in the frequency during this period. The test phase was planned to begin between week 19 and week 23, with the design/code phase completing during that period (see Figure 5.3.1). Actually, the two phases progressed concurrently through to about week 36. The increase in the frequency of waiting might

[3] Smoothers with a range of three, five, seven, nine and eleven datapoints were explored. The smaller ranges (i.e. three, five and seven datapoints) appeared to be too sensitive to the fluctuations in the evidence. The largest range (i.e. eleven datapoints) appeared to be too *in*sensitive to the fluctuations in

partly reflect the concurrent progress of these two phases. Furthermore, the test phase persists through to approximately week 51, with a sudden drop-off in testcases for weeks 50 and 51. In these two weeks, the test process area concentrated on testing the 'Gold Master' tape that had been prepared for manufacturing. Figure 5.3.3 presents indicators of an increase in project activity; these indicators beginning in week 22 and persisting through to week 49.

The increase in the frequency of waiting may also reflect the commencement of the defect-fixing process, where the test process area identify defects, returning them to the design/code process area for fixing, and subsequently waiting on the design/code process area to supply the fixes.

In sum, the frequency of waiting suggests that, for Project B, threats to the development capability increase from week 20, and that the increase in threats is due to a general increase in project activity (e.g. with more tasks, decisions etc. progressing concurrently), the concurrent progress of two phases (phases which later evidence suggests are the most fundamental phases in these two software development projects), and the inter-dependence between process areas caused by the commencement of the defect-fixing process.

A similar pattern of behaviour is also apparent for Project C, although harder to discern. For Project C, the increase in references to waiting begins in week 32 and persists through to week 54. The design/code phase was planned to complete in week 27, with the test phase commencing in that week (see Figure 5.4.1 for more information). The design/code phase actually completes around week 49, with the design/code and test phases proceeding concurrently for approximately 22 weeks, from week 27. Figure 5.4.3 shows a revised plan of the second phase of system rest commencing in week 32. As chapter five explained, development of the new product in Project C was more problematic than the enhancements to the DOS- and OS/2-based products, so the system test plan was revised to 'push back' testing the ported product until later in the test phase. This would provide the design/code process area with more time to complete the work on the new product. The increase in the frequency of waiting from week 32 may reflect the impact of the design/code process area not completing the ported product by the revised planned start of the ported product's test phase.

Figure 5.3.3 (in chapter five) presents indicators of an increase in project activity; these indicators beginning in week 24 and persisting through to week 51. As with Project B, the

---

the evidence. A range of nine datapoints appeared to provide the best smoothed representation of the

frequency of waiting may also reflect the defect-fixing process that would begin around week 27.

In sum, the frequency of waiting suggests that for Project C, like Project B, threats to the development capability increase from week 32, and that the increase in threats is due to a general increase in project activity (e.g. with more tasks, decisions etc. progressing concurrently), the concurrent progress of two phases (phases which later evidence suggests are the most fundamental phases in these two software development projects), and the inter-dependence between process areas caused by the commencement of the defect-fixing process.

As already stated, the behaviour of Project C is not as clear in its pattern as that for Project B but that a similar pattern is apparent for the two projects. Difficulties in identifying the pattern for Project C may be due to a number of causes. First, there were less status meetings for Project C so a pattern would be harder to discern. Second, the status meetings for Project C were design/code/test status meetings, whereas for Project B they were project status meetings. Finally, Project C is managed differently to Project B (see chapter five), and this may distort the pattern for Project C.

The prevalence of waiting

As explained in chapter three, the evidence collected from the two projects can be used to test one of Bradac *et al.*'s ([18]) conjectures *viz.* that waiting is more prevalent during the beginning and the end of the project. Because of the lack of evidence for the beginning of the project, only the second part of Bradac *et al.*'s conjecture can be investigated. Also, because of the focus of this investigation, the conjecture is investigated at a higher-level of the process. Stated explicitly as a hypothesis, Bradac *et al.*'s conjecture takes the following form:

$H1_{Exp}$ For the process areas of the project, waiting on blocked work is more prevalent during the end of the project than during the middle of the project.

Table 6.3.2 Mann Whitney *U* tests of hypothesis $H1_{Exp}$

Project	*p*	α	N
B	0·0008	0·001	46
C	0·002	0·01	33

tendencies within the evidence.

Table 6.3.3 Summary statistics for the middle and end of the project

Project	Project stage	Count	Median	Mode	Min	Max	Range
B	Middle	17	1	0	0	6	6
	End	29	3	1	0	6	6
C	Middle	13	0	0	0	2	2
	End	20	1.5	1	0	4	4

Table 6.3.2 presents the results of two Mann Whitney *U* tests of hypothesis $H1_{Exp}$ for Projects B and C respectively. Definitions of the beginning, middle and end of a project are provided in chapter three. For the test of $H1_{Exp}$ for Project B, 46 cases are used rather than the complete 49 cases because the first three cases occur during the beginning of the project. For the same reason only 33 cases, rather than the total 37, are used in the test for Project C. Tied values are used in both tests. See Appendix B4 for full details of the tests. As indicated in Table 6.3.2, the null hypothesis is rejected and the experimental hypothesis is retained for both tests (for Project B, $p=0{\cdot}0008$, $\alpha=0{\cdot}001$, $N=46$; for Project C, $p=0{\cdot}002$, $\alpha=0{\cdot}01$, $N=33$). Table 6.3.3 provides summary statistics for the data used in the two tests.

The two tests confirm Bradac *et al.*'s conjecture that waiting is more prevalent during the end of the project than during the middle of the project, but at a higher level of the process (Bradac *et al.* studied an individual designer whereas this investigation studied process areas and the project level). Consequently, there is some support for Bradac *et al.*'s requirement that the global level of the process be "consonant" with the local level for a reduction in waiting to reduce project duration.

Explanations for the frequency and prevalence of waiting

One possible explanation for the prevalence of waiting is that waiting *appears* more prevalent during the end of the project because there is an increase in *reporting* waiting during the end of the project: as the balance between capability and workload increasingly fluctuates so the reporting of waiting increases, even though the underlying waiting does not increase in its frequency. A second possibility is that there is an interaction between an actual increase in waiting and an increase in the reporting of waiting. These two explanations are both consistent with the increase in project activity (tactics of management), discussed in chapter five.

Another possible explanation for the prevalence of waiting during the end of the project is that the number of concurrent phases increases and, as a result, the middle of the

project merges with the end of the project. (In chapter three, the middle of the project is broadly defined as the design/code phase and the end of the project is broadly defined as the test phase.) As discussed in chapter five, the design/code phases for both projects completes later than planned and there were a number of weeks in which the design/code phases and the test phases progressed in parallel. It might be that with the concurrency of phases, waiting increases. An investigation of the effect of concurrent phases on the prevalence of outstanding work stands as one opportunity for further research.

6.4 The types of waiting and their frequencies

A closer examination of the various types of waiting (strictly, the different types of work for which process areas are waiting) allows a replication of another of Bradac *et al.*'s observations (i.e. the types of waiting and their frequencies), and provides further insights into the roots of process problems and threats to capability.

Two classifications were used to examine the types of waiting. The first classification is taken from Bradac *et al.* ([18]; see chapter two for further information). The second classification was generated inductively from the evidence.

Comparison of the types of waiting using Bradac *et al.*'s classification

Table 6.4.1 Comparison of types of waiting (using Bradac *et al.*'s classification)

	Project B		Bradac *et al.*	Project C	
Category	Count	% total	% waiting	% total	Count
Other	53	51.4	66.7	42.9	18
Review	0	0.0	15.1	4.8	2
Expert	1	1.0	5.1	4.8	2
Laboratory	0	0.0	4.5	0.0	0
Documentation	3	2.9	3.9	0.0	0
Software	45	43.7	3.1	45.2	19
Hardware	1	1.0	1.6	2.3	1
Total	103	100.0	100.0	100.0	42

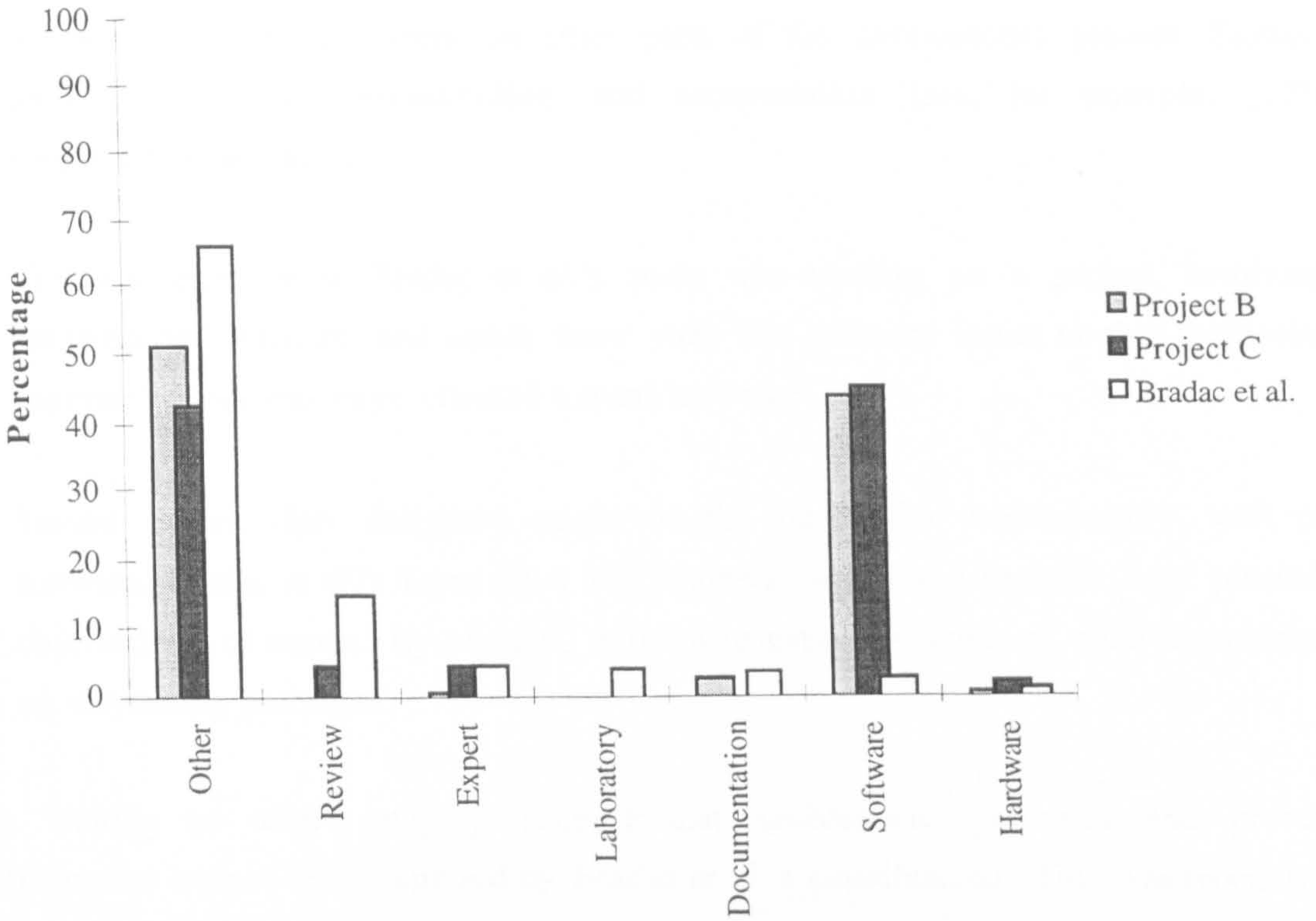


Figure 6.4.1 Comparison of types of waiting (using Bradac *et al.*'s classification)

Table 6.4.1 and Figure 6.4.1 compare the results of Project B, Project C and Bradac *et al.*'s lead engineer, based on the classification system used by Bradac *et al*. The evidence collected from Projects B and C only include references to waiting, whereas Bradac *et al.* included references to working the process aswell as references to waiting. Consequently, Bradac *et al.*'s percentages of waiting have been adjusted, for the purposes of comparison, by removing the influence of working the process (see chapter three for more information).

The high proportion of references to waiting on software is not surprising when one considers that these projects are developing software products. This suggests that the software production processes (e.g. design and code) are potentially the most problematic. Interestingly, Bradac *et al*. found a low proportion of references to waiting on software. Possible explanations for this might be:

1. Lead engineers (the focus of Bradac *et al.*'s study) are more 'insulated' from other parts of the development process, and consequently may be more 'autonomous' in their own development processes. Thus, they do not wait on the availability of software because they are not dependent on that software. But at higher levels of the

project, such as the process areas and the project itself, project members become less insulated and more dependent on other parts of the development process. Parnas' prescription of information-hiding and encapsulation (see, for example, [19]) supports this argument.

2. The lead engineer in Bradac *et al.*'s study was working on a project involving hardware and software, and either there were less software issues or that particular engineer's work was more oriented toward hardware.

3. Testers, rather than designers, might be the ones who predominantly wait on software. Bradac *et al.*'s focus on a lead engineer, who was a designer, may preclude observations of testers. By contrast, this investigation was able to analyse evidence on the testing processes.

The waiting on other category suggests that problematic processes and process inefficiencies are not being captured by Bradac *et al.*'s classification. This was recognised by Bradac *et al.*

Comparison of the types of waiting using the alternative classification

Table 6.4.2 Comparison of types of waiting (using the alternative classification)

	Project B		Project C	
Category	**Count**	**%**	**%**	**Count**
Decision	44	42.7	16.7	7
Defect/Fix	27	26.2	33.3	14
Code	18	17.5	7.1	3
Other	5	4.8	23.8	10
Information	4	3.9	11.9	5
Resource	3	2.9	0.0	0
Unknown	2	1.9	7.1	3
Total	103	99.9	99.9	42

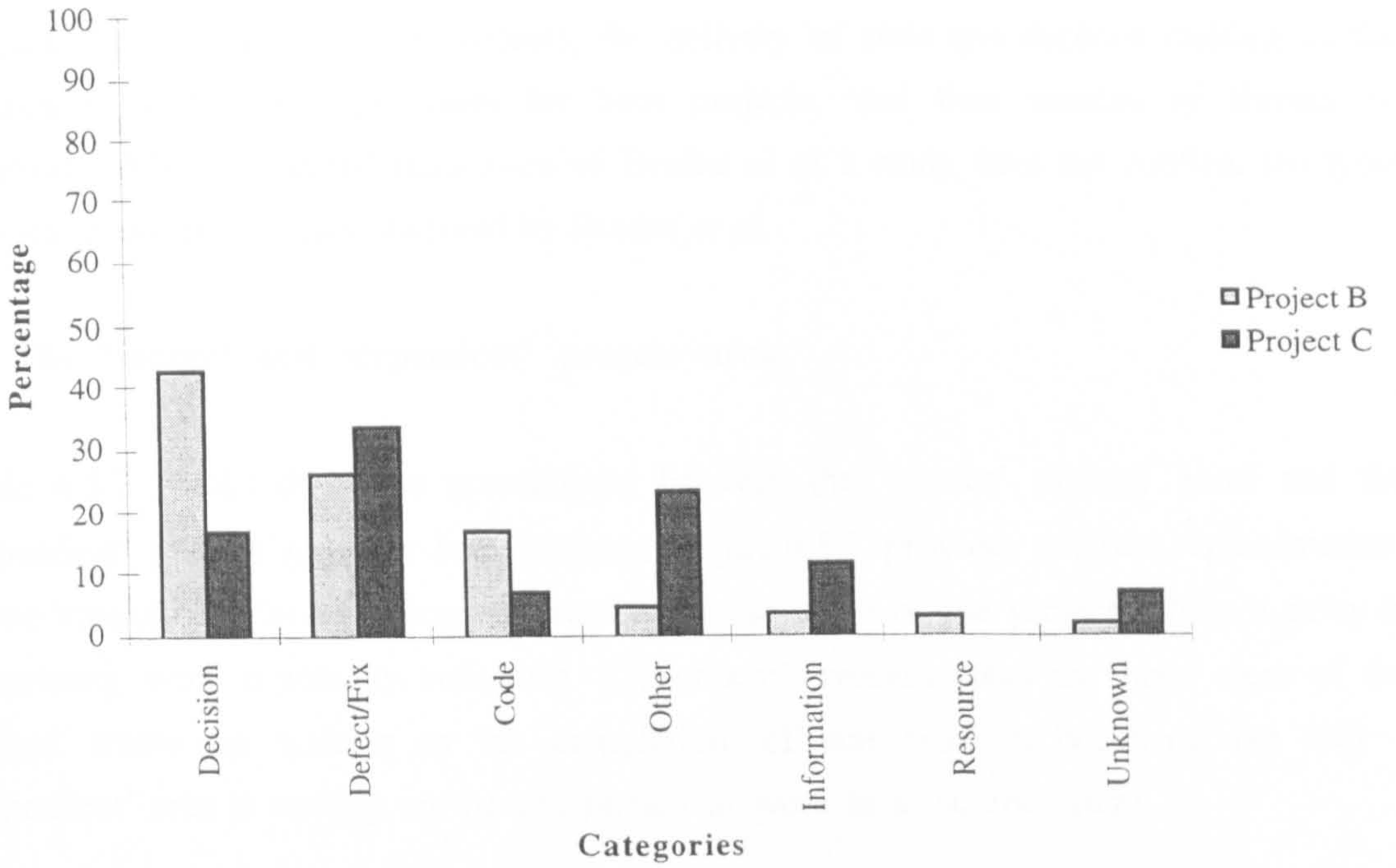


Figure 6.4.2 Comparison of types of waiting (using the alternative classification)

Table 6.4.2 and Figure 6.4.2 compare the results of Project B and Project C, based on the alternative classification for the two projects. The table and figure indicate that, with the exception of the Defect/Fix category of waiting, there is little consistency between the two projects. The Defect/Fix and Code categories in Table 6.4.2 are conceptually similar to the waiting on software category in Table 6.4.1 (and suggest the composition of the category 'waiting on software'). This suggests that process areas within Projects B and C often wait on either software or fixes to software defects. In turn, this suggests that the defect process and the coding process are either problematic processes in themselves or are impacted by problematic processes.

As discussed in chapter two, Bradac *et al.* found that the Other category in their classification system (see Table 6.4.1) actually consisted of a variety of Not working categories i.e. Training, Other Assignments, Vacation, Weekend, and Other. In classifying the waiting evidence from Projects B and C, it became clear that those items of waiting classified as Waiting on other, using Bradac *et al.*'s classification, were subsequently classified as Decision, using the alternative classification. This suggests an empirical difference in the waiting on other category between this investigation and Bradac *et al.*'s investigation. The lack of references to the Not working categories may be due to the different focus of this study i.e. on process areas and the project, rather than the local process of an individual designer.

Overall, this analysis points to defects, the delivery of code and decision-making as the potentially problematic processes for both projects, and thus sources of threats to capability. Also, the partial replication of Bradac *et al.*'s study does not confirm the types of waiting and frequencies observed by Bradac *et al.*

6.5 The 'source' and 'dependent' process areas

Table 6.5.2 breaks down the associations between the 'source' process areas and the 'dependent' process areas for both projects. Figure 6.5.1 provides a visual representation of the breakdown. 'Source' process areas are those areas of the project where a delay in completing work is actually occurring. 'Dependent' process areas are those areas of the project where the waiting on the completion of that work is occurring (so that a 'dependent' area is waiting on the completion of work in a 'source' area).

Table 6.5.1 Significant values in Table 6.5.2

Project	Groups of cells	Number of cells	Total count	Mean	P≤0.05	P≤0.01	P≤0.001
B	'Internal' cells	56	82	1.46	7	8	10
B	Column totals	7	82	11.71	21	23	26
B	Row totals	8	82	10.25	19	21	24
C	'Internal' cells	56	31	0.55	5	6	7
C	Column Totals	7	31	4.43	11	12	14
C	Row totals	8	31	3.87	10	11	13

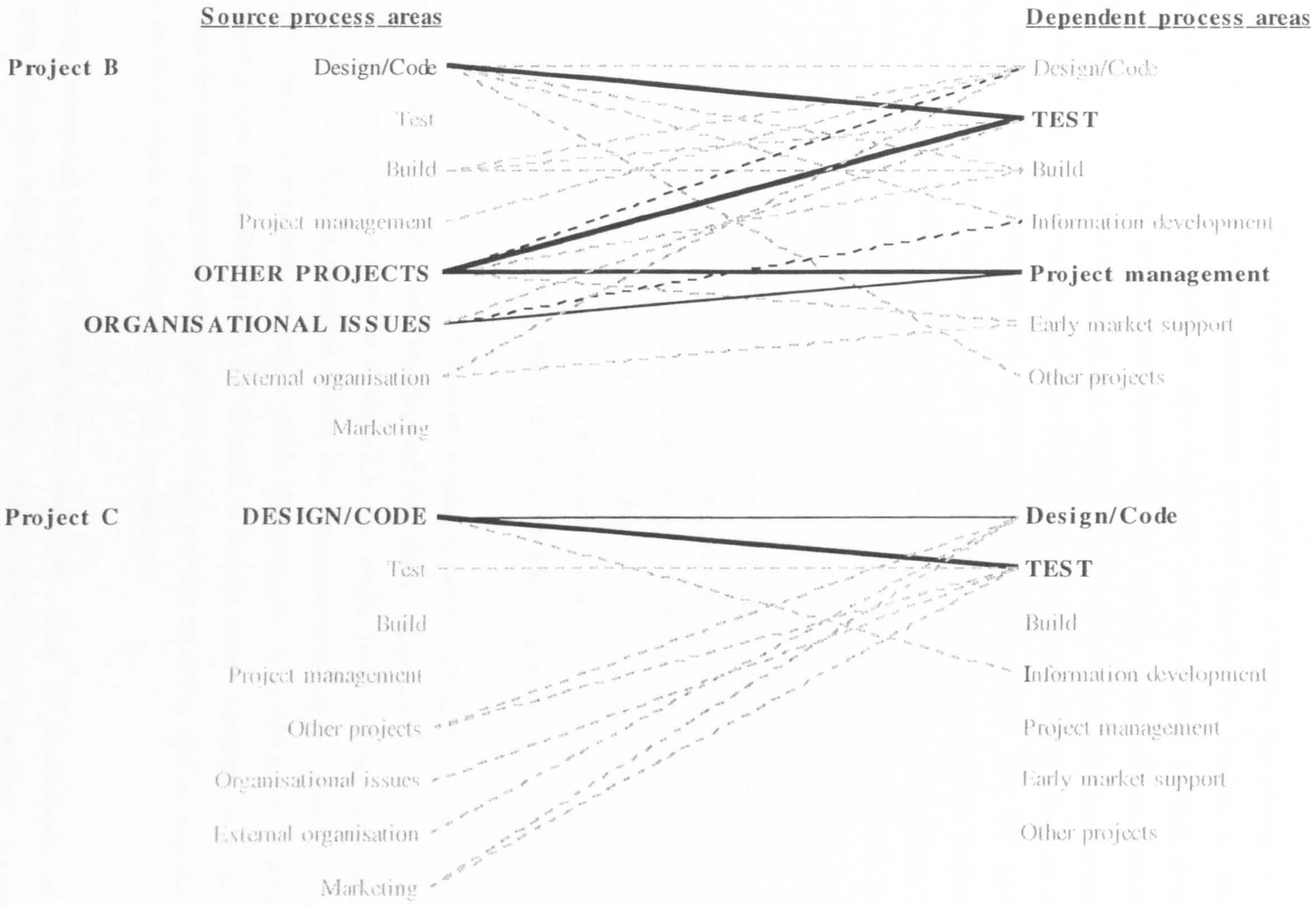
Table 6.5.1 presents the probability thresholds at which values in Table 6.5.2 become significant. Probabilities were calculated to a 95% confidence interval. A full explanation of how the probabilities were calculated is provided in Appendix A0. The Number of cells and Total count recorded in Table 6.5.1 is smaller than the actual number of cells and actual total count in Table 6.5.2 because the Unknown category has been removed from the calculations.

In Figure 6.5.1, thick, black solid lines represent extremely significant associations (i.e. P≤0.001). Solid black lines represent very significant associations (i.e. P≤0.01). Broken black lines represent significant associations (i.e. P≤0.05). Grey, broken lines represent non-significant associations. Names of process areas that are emboldened and capitalised represent those process areas with an extremely significant number of references. Names of process areas that are emboldened represent those process areas with a very significant number of references. Names of process areas that are in normal text represent those process areas with a significant number of references. Names of process areas that are in grey text are not significant.

Table 6.5.2 Breakdown of the 'source' and 'dependent' process areas

	Dependent process area																	
	Build		Other projects within the laboratory		Design / Code		Early market support		Information develop-ment		Project management		Test		Unknown		Total	
Source process area	B	C	B	C	B	C	B	C	B	C	B	C	B	C	B	C	B	C
Build	1		0		1	1	0		0	0	0		5	0	0	0	7	1
Design/Code	1		1		3	6	0		3	4	0		11	10	0	0	19	20
Test										0				1		0		1
Project management	0		0		1	0	0		0	0	0		0	0	0	0	1	0
Other projects within the organisation	2		0		7	3	1		0	0	8		13	1	2	0	33	4
Organisational issues	0		0		0	0	0		7	0	16		1	1	2	0	26	1
External organisation	0		0		1	2	1		0	0	0		0	0	0	0	2	2
Marketing						1				0				1		0		2
Unknown	4		1		1	0	1		0	0	0		2	5	6	6	15	11
Total	8		2		14	13	3		10	4	24		32	19	10	6	103	42

Figure 6.5.1 Associations between the 'source' and 'dependent' process areas



It is clear that for both projects, there are complex relationships between the source and dependent process areas. It is also clear that there are few similarities between the two projects, although there is some indication that the design/code and test process areas are significant for both projects.

For Project B, it is clear that external process areas have a strong association with internal process areas, whereas such associations are not apparent for Project C. This suggests that Project B's internal process areas frequently wait on external process areas, whereas Project C's internal process areas frequently wait only on other internal process areas. This difference between Project B and Project C is consistent with the contrasting strategic values of the two products, and with the fact that Project B is dependent on an external project delivering designs and code to Project B (see chapter five for more information). The difference between Project B and Project C also suggests that Project B would have greater difficulty managing the project and improving the project's development processes, because some of the problematic processes are beyond the control of Project B's management.

The number and variety of associations, regardless of whether these associations are individually significant, clearly indicates that there are multiple associations across multiple process areas. This suggests multiple dependencies between process areas. These dependencies may be manageable using a plan consisting of sequential, discrete phases. Alternatively, these dependencies suggest that a project consists of a number of processes iterating through a number of process areas (*cf.* [92]; see also chapter two).

Chapter five discussed how Project C's external re-plan occurred officially because of the introduction of new year-2000 requirements, and that these requirements were imposed by the organisation on the project. In Table 6.5.2, one might expect to find references concerning the year-2000 requirements within the 'Organisational issues' process area. Surprisingly, there are very few references within this category. It is difficult to explain this discrepancy. One possibility is that the absence of project status meetings in Project C meant that the management of organisational issues was not recorded, and consequently evidence on these issues is not available to this investigation.

A potentially rewarding exercise would be to examine the 'flow' of associations, as a process area alternates between being a 'source' process area and a 'dependent' process area. One example is the design/code process area for Project B. There is a significant association between Other projects, as the source process area, and Design/Code as the dependent process area. There is then an extremely significant association between Design/Code and Test. Such analysis might suggest possible knock-on effects from Other

projects through Design/Code to Test. Another example, again with Project B, suggests feedback relationships: the Design/Code and Build process areas are both source and dependent process areas with regards to each other. Due to practical limitations, these kinds of analyses are beyond the scope of this investigation, and stand as opportunities for further research.

6.6 Process areas and types of waiting

Tables 6.6.3 and 6.6.4 break down the types of waiting for the 'source' and 'dependent' process areas for the two projects. Figures 6.6.1 and 6.6.2 provide visual representations for the two tables. The types of waiting form the basis for the relationships between source and dependent process areas, as discussed in the preceding section.

Table 6.6.1 Significant values in Table 6.6.3

Project	Groups of cells	Number of cells	Total count	Mean	P≤0.05	P≤0.01	P≤0.001
B	'Internal' cells	40	83	2.075	8	9	11
B	Row totals	8	83	10.375	26	28	31
B	Column totals	5	83	16.6	19	21	23
C	'Internal' cells	40	25	0.625	5	6	7
C	Row totals	8	25	3.125	11	12	14
C	Column Totals	5	25	5	9	10	12

Table 6.6.2 Significant values in Table 6.6.4

Project	Groups of cells	Number of cells	Total count	Mean	P≤0.05	P≤0.01	P≤0.001
B	'Internal' cells	35	92	2.628	9	10	12
B	Row totals	7	92	13.14	29	31	34
B	Column totals	5	92	18.4	23	25	27
C	'Internal' cells	35	28	0.8	5	6	7
C	Row totals	7	28	4	12	13	≥15
C	Column Totals	5	28	5.6	10	11	13

Tables 6.6.1 and 6.6.2 present the probability thresholds at which values in Tables 6.6.3 and 6.6.4 become significant. Probabilities were calculated to a 95% confidence interval. A full explanation of how the probabilities were calculated is provided in Appendix A0. The Number of cells and Total count recorded in Tables 6.6.1 and 6.6.2 is smaller than the actual number of cells and actual total count because the Unknown category has been removed from the calculations.

Table 6.6.3 Types of waiting and 'source' process areas

Type of waiting	Source process area: Build B	Build C	Design / Code B	Design / Code C	Test B	Test C	Market-ing B	Market-ing C	Organ-isational issues B	Organ-isational issues C	Other projects within the organisation B	Other projects within the organisation C	Project management B	Project management C	External organ-isation B	External organ-isation C	Unknown B	Unknown C	Total B	Total C
Code	5	0	8	3	0		0		0	0	3	0	0		1	0	1	0	18	3
Decision	0	0	1	1	0		1		25	0	14	2	0		1	2	3	1	44	7
Defect/Fix	1	1	6	9	0		0		0	0	14	1	0		0	0	6	3	27	14
Information	0	0	4	3	0		1		0	0	0	1	0		0	0	0	0	4	5
Other	1	0	0	4	1		0		1	1	2	0	1		0	0	0	4	5	10
Resource	0	0	0	0	0		0		0	0	0	0	0		0	0	3	0	3	0
Unknown	0	0	0	0	0		0		0	0	0	0	0		0	0	2	3	2	3
Total	7	1	19	20	1		2		26	1	33	4	1		2	2	15	11	103	42

Table 6.6.4 Types of waiting and 'dependent' process areas

Type of waiting	Dependent process areas: Build B	Build C	Design / Code B	Design / Code C	Test B	Test C	Project management B	Project management C	Other project (within laboratory) B	Other project (within laboratory) C	Early market support B	Early market support C	Information development B	Information development C	Unknown B	Unknown C	Total B	Total C
Code	2		1	2	13	0	0		0		2		0	1	0	0	18	3
Decision	2		7	6	1	0	23		0		1		7	0	3	1	44	7
Defect/Fix	4		2	2	18	12	0		1		0		0	0	2	0	27	14
Information	0		0	1	0	1	0		1		0		3	3	0	0	4	5
Other	0		3	2	1	6	1		0		0		0	0	0	2	5	10
Resource	0		0	0	2	0	0		0		0		0	0	1	0	3	0
Unknown	0		1	0	1	0	0		0		0		0	0	0	3	2	3
Total	8		14	13	36	19	24		2		3		10	4	6	6	103	42

Figure 6.6.1 Types of waiting and 'source' process areas

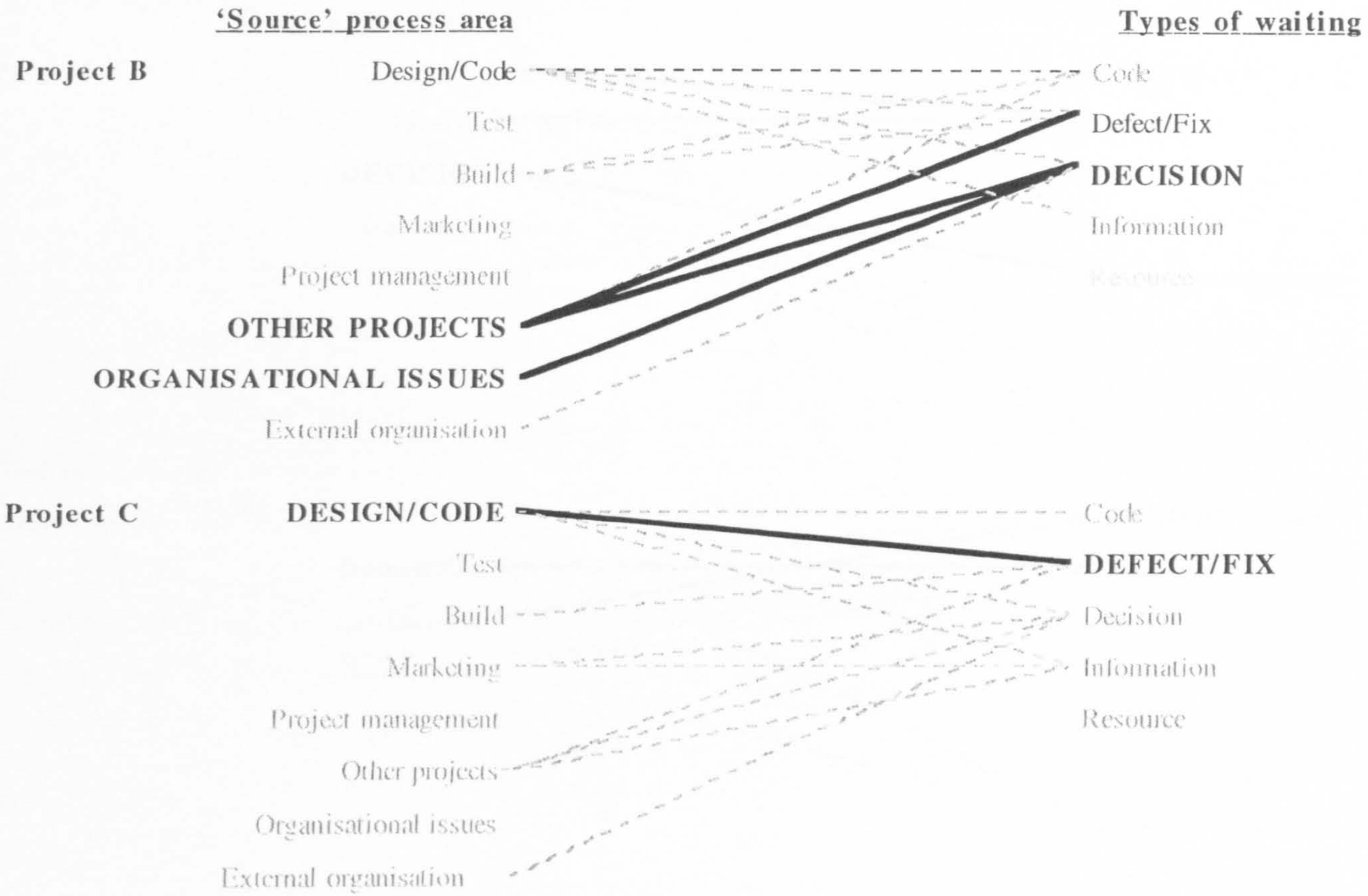
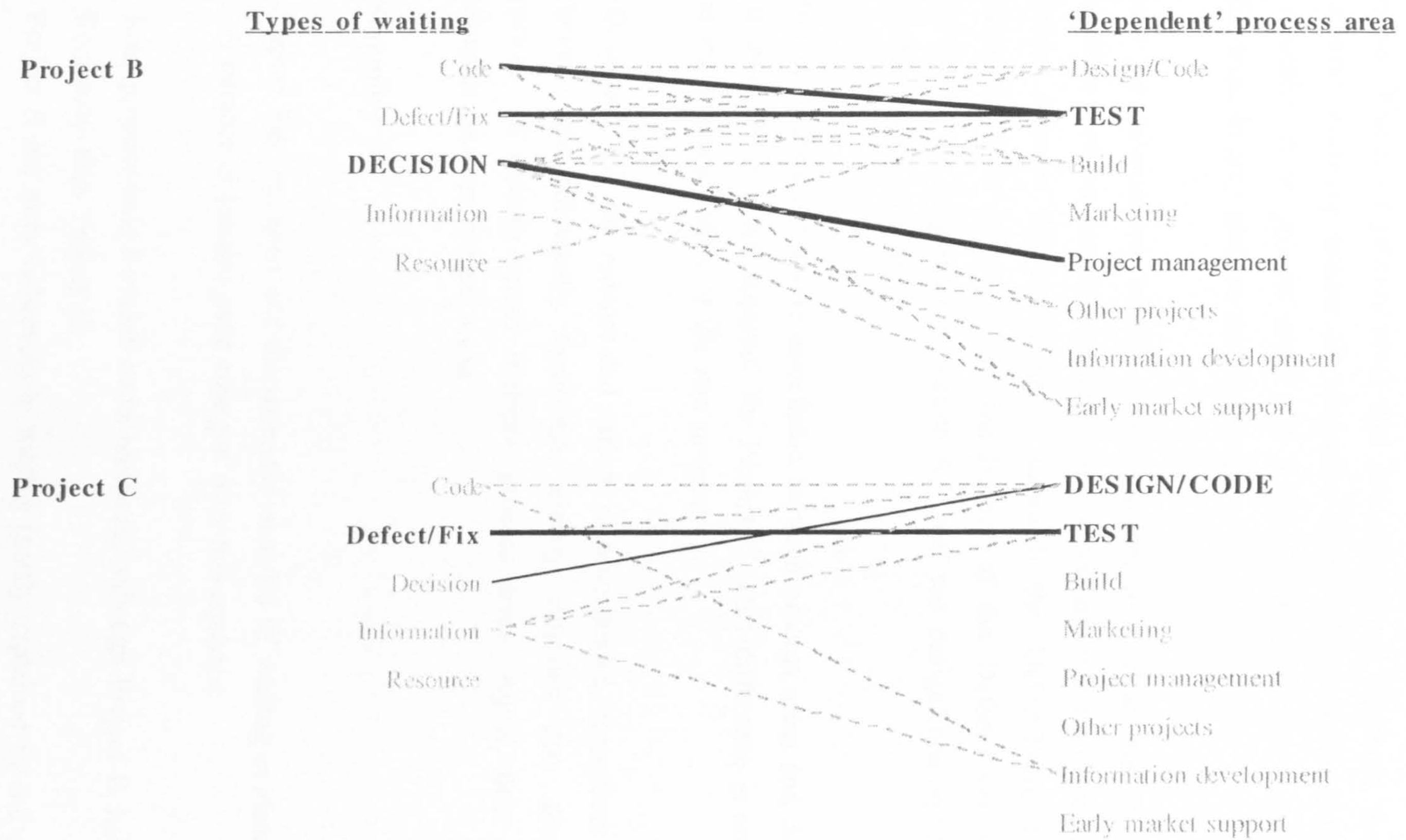


Figure 6.6.2 Types of waiting and 'dependent' process areas



For Figures 6.6.1 and 6.6.2, thick, black solid lines represent extremely significant associations (i.e. P≤0.001). Solid black lines represent very significant associations (i.e. P≤0.01). Broken black lines represent significant associations (i.e. P≤0.05). Grey, broken lines represent non-significant associations. Names of process areas that are emboldened and capitalised represent those process areas with an extremely significant number of references. Names of process areas that are emboldened represent those process areas with a very significant number of references. Names of process areas that are in normal text represent those process areas with a significant number of references. Names of process areas in grey text are not significant.

As with the associations between process areas, it is clear that there are complex relationships between process areas and types of waiting. It is also clear that there are few similarities between the two projects, although the Defect/Fixes type of work is significant for both projects. The significance of the Defect/Fixes type of work is consistent with the suggestion, in section 6.5, that the design/code and test process areas are significant process areas.

For Project B, there is a strong association between process areas and decisions, although such an association is not apparent for Project C. This difference is consistent with the contrasting strategic values of the two projects.

As with section 6.5, the number and variety of associations, regardless of whether these associations are individually significant, clearly indicates that there are multiple associations of waiting across multiple process areas. Again, this suggests multiple dependencies between process areas.

6.7 Summary

This chapter has presented and discussed the analysis of waiting evidence for Projects B and C. A number of insights have emerged from this analysis:

1. Both projects held frequent status meetings, although Project B held meetings more frequently than Project C.
2. Project B has more references to waiting (partly explained by the greater number of status meetings) and a more noticeable increase in waiting.
3. For both projects, the increase in the frequency of waiting might be partly explained by the concurrent progress of the design/code and test phases, by the commencement of the defect-fixing process, and/or by the increase in project

activity on the projects. This suggests that threats to the capability of the development process are due to:

- a general increase in project activity
- the concurrent progress of the design/code and test phases
- the inter-dependence between design/code and test process areas, due to the nature of the defect-fixing process.

4. For both projects, and at the level of process areas of the project, waiting is more prevalent during the end (the test phase) of the project than during the middle (design/code phase) of the project. This complements the research of Bradac *et al.*
5. For both projects, there is a relatively high proportion of references to waiting on either code or fixes to defects in code. There is little consistency between the two projects in terms of other types of waiting.
6. There are clearly complex relationships between the process areas within each of these two projects, and it is difficult to establish clear similarities between the two projects. A possible similarity across the two projects is the prominence of the design/code and test process areas, both in terms of these two process areas being significant source or dependent process areas, and in terms of the significant associations between these two process areas.
7. A clear difference between the two projects is the influence of external process areas on the projects:
   - For Project B, the prominent source process areas are both external and internal to the project, and the prominent dependent process areas are, naturally, internal to the project.
   - For Project C, the prominent source process areas are only internal to the project, and the prominent dependent process areas are, naturally, internal to the project.
8. The multiple associations across multiple process areas suggests that the two projects each have multiple dependencies between process areas, and also that each of the projects have a number of processes iterating through a number of process areas.
9. There are also complex relationships between process areas and types of waiting although, again, similarities are not apparent.
10. For Project B only, Decisions are a significant type of waiting. This is consistent with the relatively high strategic value of Product B to the organisation

Chapter 7 The progress of work

7.1 Introduction

This chapter presents and discusses the analysis of the progress of work evidence for Projects B and C. The evidence is used in three ways. First, to provide insights into the nature of process areas within a project. Second, to provide evidence relating to the model of capability (recall from chapter four that poor progress reflects a preceding imbalance between workload and capability). Third, to provide evidence to test Bradac *et al.*'s conjecture that waiting is more prevalent during the end of the project than during the middle of the project.

The specific questions investigated in this chapter are:

- What is the frequency of references to progress of work?
- What is the prevalence of poor progress of work over the duration of the project? This is intended as a complementary test of Bradac *et al.*'s conjecture.
- What are the different types of progress of work, and what are their frequencies?
- What is the breakdown of the types of progress against functional areas of the project?
- What are the causes of poor progress?
- What are the causes of good progress?

Details on the methods used to collect, organise and analyse the progress of work evidence are provided in chapter three.

7.2 The frequency and prevalence of progress of work

The frequency of progress of work

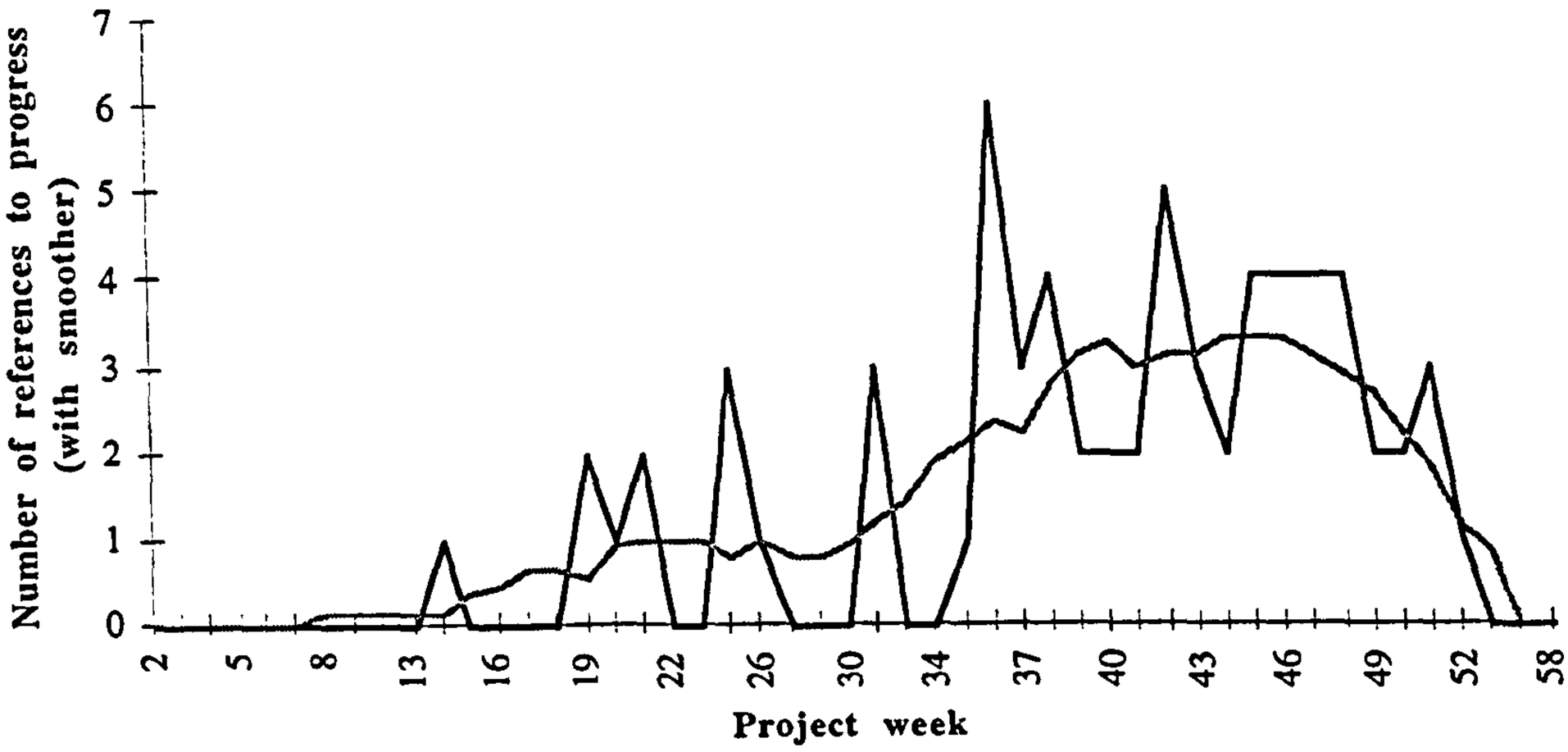


Figure 7.2.1 References to the progress of work for Project B

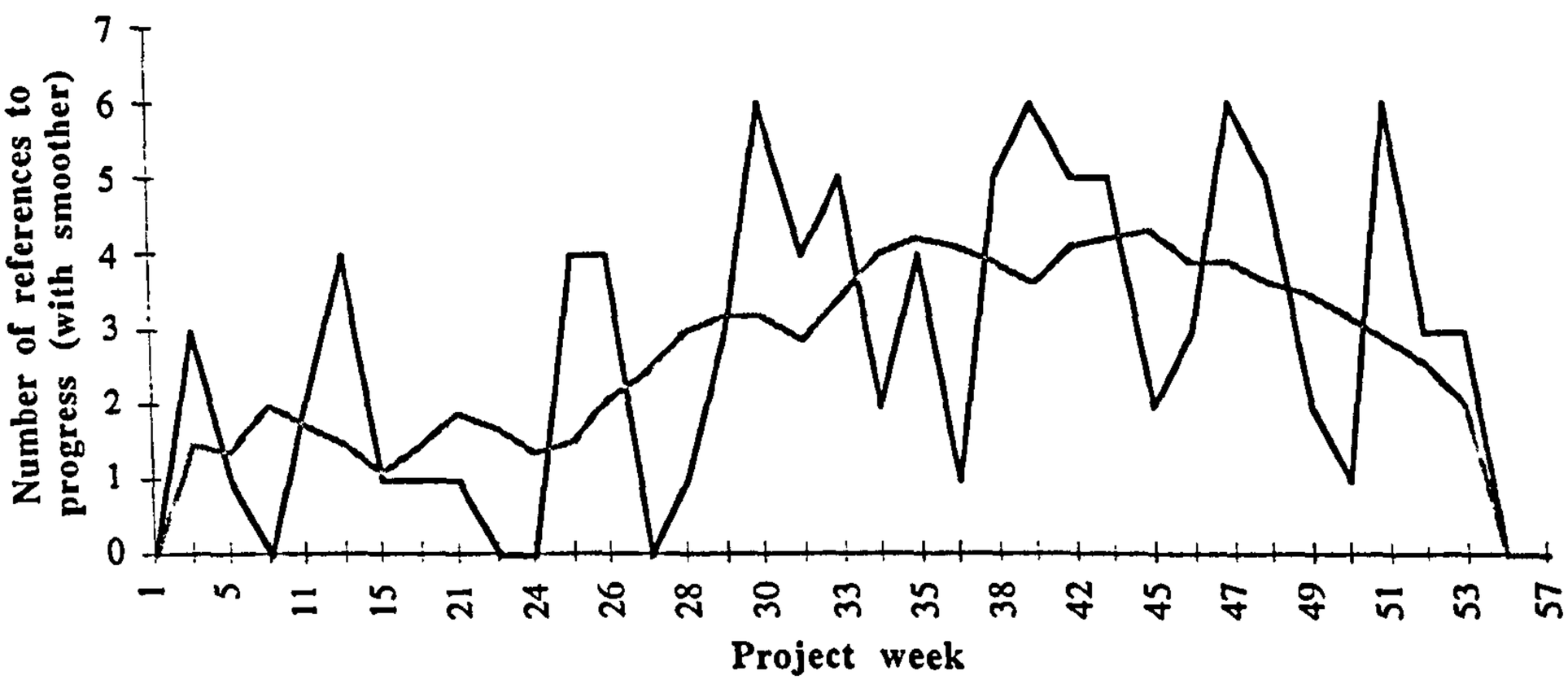


Figure 7.2.2 References to the progress of work for Project C

Table 7.2.1 Summary statistics for the frequency of progress of work

Project	Count	Mean	Median	Mode	Min	Max	Range	Total
B	49	1.4	1	0	0	6	6	67
C	37	2.7	3	0	0	6	6	99

Figures 7.2.1 and 7.2.2 present lineplots of all references to the progress of work for Projects B and C. The lineplots include a smoother, calculated as a moving average with a range of nine datapoints. (Precise values are presented in Appendix B5.) Only those

weeks where a status meeting occurred are included in the lineplots. Table 7.2.1 provides summary statistics to complement the two figures.

For Project B, Figure 7.2.1 indicates that the reporting of progress fluctuates, with a noticeable increase in reporting progress in the later weeks of the project i.e. from week 35 onwards. The lack of reports of progress in the first 34 weeks of the project is interesting: a considerable amount of work would have occurred in these weeks and yet there are almost no references to the progress of this work. This suggests that reports of progress are not only a function of progress itself.

Comparison of Figure 7.2.1 with the figures presented in chapter five reveal interesting patterns. The sudden increase in references to the progress of work in week 35 is one week before the planned completion of the functional verification phase (planned to complete in week 36), and about one week before the actual completion of the design/code phase. With the exception of the 'Design complete' checkpoint, all the major checkpoints of the project occurred after week 35, with the first of these checkpoints ('FV complete') planned to occur in week 36. It is likely that the series of checkpoints introduce pressure into the project to complete work by the time of the checkpoint, and this may lead to an increase in the reporting of progress. Similarly, most of the indicators of project activity occurred after week 35. This may also affect the reporting of progress. The drop-off in the number of references to the progress of work in week 52 is the week that the product was actually delivered (even though testing continued for about six further weeks).

For Project C, Figure 7.2.2 indicates that the reporting of progress also fluctuates, with an increase in reporting in the second half of the project i.e. from week 28 onwards. Like Project B, there is a noticeable trend for more references to the progress of work later in the project. Once again, given the amount of work that would have been conducted in the first half of the project, there are relatively few references to the progress of work (although more references than Project B) in the first 28 weeks of the project.

Comparison of Figure 7.2.2 with the figures presented in chapter five reveal interesting patterns. Week 28 is one week after the planned completion of the design/code phase and one week after the actual start of the test phase. Week 28 is one week before the re-planned and actual start of testing the product on the new operating system. Finally week 28 is the week that a member of the project stated: "... this coincides with probably the busiest time of Development". The drop-off in the number of references to waiting in week 54 is one week before the product is dispatched to manufacturing.

The prevalence of the poor progress of work

Chapter six examined the prevalence of waiting between the middle and end of the project, as a test of one of Bradac *et al.*'s conjectures ([18]). A similar test can be conducted for the poor progress of work. Explicitly:

$H2_{Exp}$ For the process areas of the project, poor progress of work is more prevalent during the end of the project than during the middle of the project

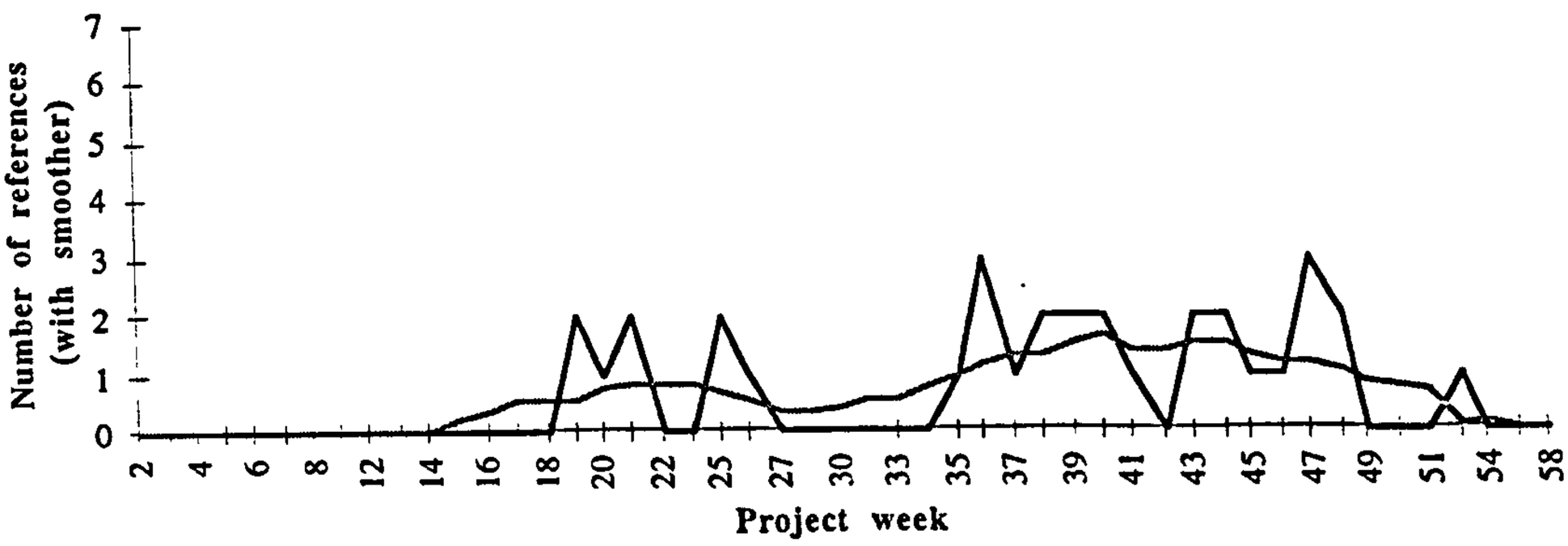


Figure 7.2.3 Frequency of poor progress for Project B

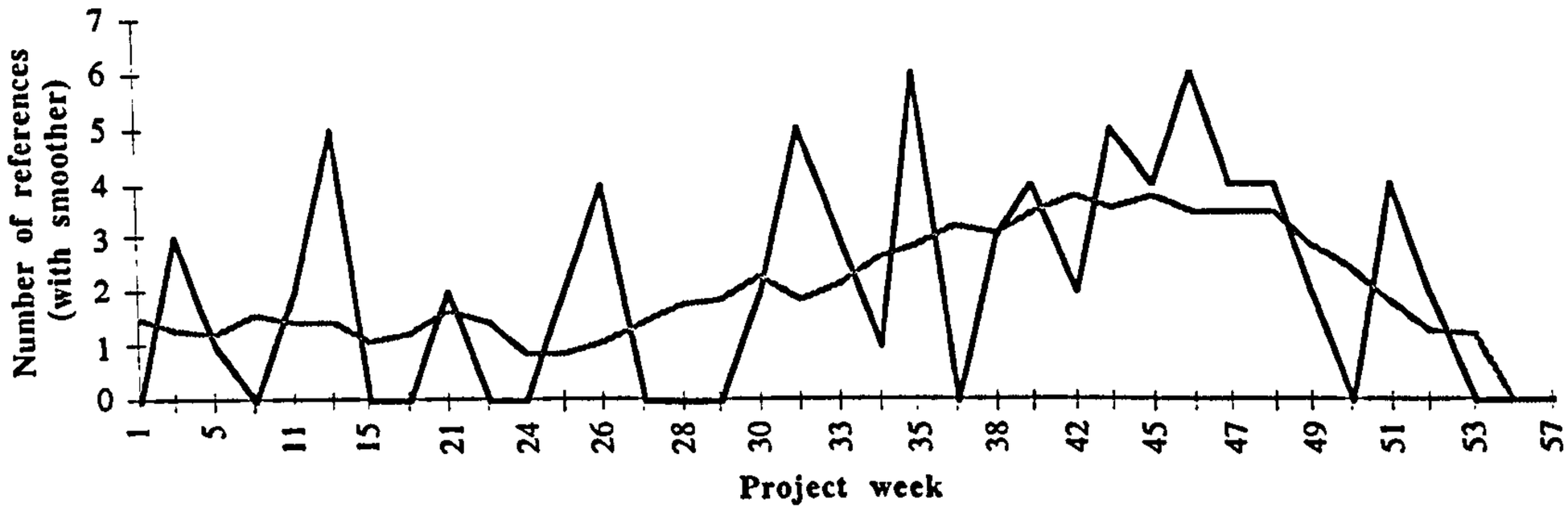


Figure 7.2.4 Frequency of poor progress for Project C

Figures 7.2.3 and 7.2.4 present lineplots of references to the *poor* progress of work for Projects B and C. Only those weeks where a status meeting occurred are included in the lineplots. For each lineplot, a smoother has been included, calculated as a moving average with a range of nine datapoints. The figures suggest that poor progress is more prevalent at the end of the projects than during the middle of the projects. The weeks of most reports of progress of any kind (see Figures 7.2.1 and 7.2.2) are also the weeks of the most reports of poor progress, as given in Figures 7.2.3 and 7.2.4. This raises two speculations. First, that poor progress dominates reports of progress. Section 7.3 shows

that reports of poor progress account for almost 50% of all reports. Second, it is only during periods of poor progress that representatives of process areas report on any kind of progress (good, reasonable or poor).

Table 7.2.2 Results of the Mann Whitney *U* tests of hypothesis $H2_{Exp}$

Project	*p*	α	N
B	0·0091	0·05	46
C	0·0238	0·05	33

Table 7.2.3 Summary statistics for the tests of poor progress of work

Project	Stage of project	Count	Median	Mode	Min	Max	Range
B	Middle	17	0	0	0	2	2
	End	29	1	0	0	3	3
C	Middle	13	0	0	0	5	5
	End	20	3	0	0	6	6

Table 7.2.2 presents the results of two Mann Whitney *U* tests of hypotheses $H2_{Exp}$. A definition of the beginning, middle and end of the project is given in chapter three. As with the waiting evidence, 46 cases are used rather than the complete 49 cases for the test of Project B, and 33 cases, rather than the total 37, are used in the test for Project C. Tied values are used in both tests. See Appendix B5 for full details of the tests. As indicated in Table 7.2.2, the null hypothesis is rejected and the experimental hypothesis is retained for both tests (for Project B, p=0·0091, α=0·05, N=46; for Project C, p=0·0238, α=0·05, N=33). Table 7.2.3 presents summary statistics of the data used in the two tests.

If the integrated model presented in chapter four is valid, then the confirmation of hypothesis $H2_{Exp}$ provides complementary support for hypothesis $H1_{Exp}$ and Bradac *et al.*'s conjecture that waiting is more prevalent during the end of the project than during the middle of the project. This is because poor progress results in outstanding work, which results in waiting.

Explanations for the frequency and prevalence of progress

There are a number of possible explanations for the frequency of references to progress of work, and the prevalence of *poor* progress. First, it might be that progress is not reported in the earlier stages of the project because it is difficult to assess progress in these stages (see, for example, [1, 2, 19]). Chapter five showed that most of the major milestones in the two projects did not occur until the second half, or even the last quarter,

of the projects. The scheduling of these milestones is consistent with the frequency of reporting progress.

Second, managers may concentrate on reporting exceptions. It may be that during the first half of these two projects the projects progressed as planned and so managers do not report on progress. As the project becomes more troublesome the managers report on both 'good' and 'bad' exceptions ('good exceptions' may occur where a project is consistently progressing poorly and then some good progress occurs). In the feedback workshop for Project B (B.FW.001), the Project Leader explained that he wanted representatives of the process areas to report poor progress, and that he wasn't interested in good progress. This explanation is consistent with a speculation raised earlier, which is that the weeks with the most reports of progress were also the weeks with the most reports of poor progress. This explanation might be affected by the first explanation i.e. as progress is difficult to assess, the managers assume that the project is progressing to plan. Abdel-Hamid ([2]) argues that reports of actual progress often simply reflect the planned progress because a more accurate assessment of actual progress is not possible.

Third, it may be that during the status meetings, the managers reported good, reasonable and poor progress, but that only reports of poor progress were actually recorded in the minutes.

Fourth, as the project approaches its planned conclusion, so the urgency of the project increases and so managers report on progress more often. The tail-off in reporting progress at the end of the project might be explained by the fact that the product is delivered to manufacturing a couple of weeks prior to the product delivery date.

Finally, it may be that the concurrent phases of the project affect the amount of poor progress on the project. An investigation of the effect of concurrent phases on the prevalence of outstanding work stands as one opportunity for further research.

7.3 The different types of progress of work, and their frequencies

Table 7.3.1 Types of progress of work for Projects B and C

Type of progress	Project B Count	Project B %	Project C %	Project C Count
Good progress	11	16.4	24.2	24
Reasonable progress	8	11.9	2.0	2
Slow progress	19	28.7	23.3	23
No progress	13	19.4	22.2	22
Other types	16	23.6	28.3	28
Total	67	100.0	100.0	99
Poor progress	32	48.1	45.5	45

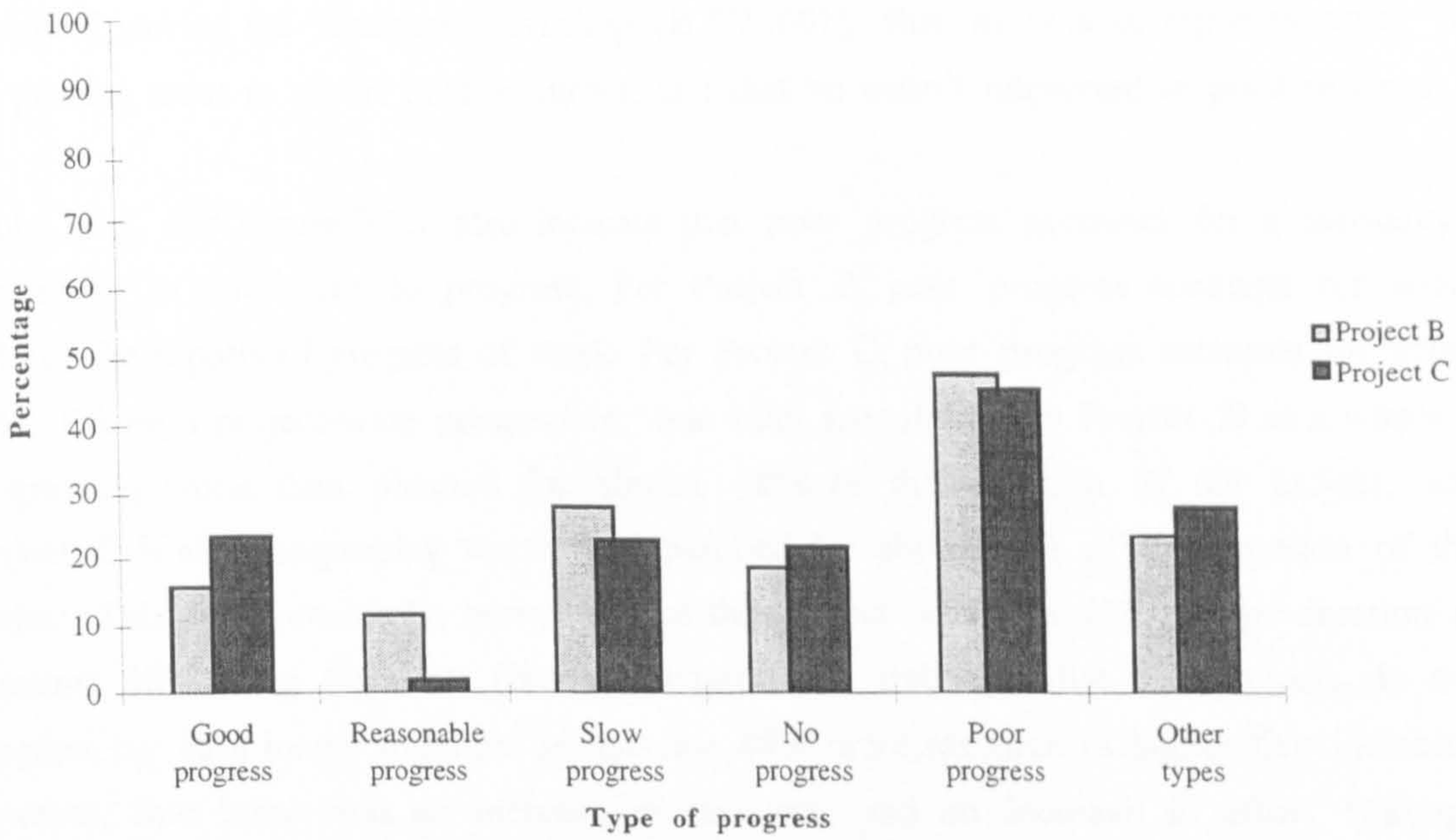


Figure 7.3.1 Types of progress for Projects B and C

Table 7.3.1 and Figure 7.3.1 break down the common types of progress for Projects B and C. Initial types of progress were first identified separately for each project, and then compared to establish common types across the two projects. Poor progress is an aggregate type, formed from the slow progress and no progress types.

An assumption is made that the first four types in Table 7.3.1 (i.e. good progress, reasonable progress, slow progress and no progress) constitute an ordinal scale of measurement, with reasonable progress being treated as an indicator that progress in a process area is approximately that which is intended. Phrased another way, reasonable progress indicates that workload and capability are balanced (see chapter four.) This assumption underpins the following logic:

Good progress is considered to be better progress than intended.
Reasonable progress is considered to be approximately that which was intended.
Slow progress is considered to be worse progress than intended, but better than No progress.

It is clear, particularly from the figure, that the two projects are similar in their reporting of progress, and that overall there is more reporting of poor progress (i.e. no and slow progress) than good or reasonable progress. There is some suggestion, however, for a bi-modal distribution, with the emphasis on reporting either good or bad progress (note the low reporting of reasonable progress). As already noted, the Project Leader for Project B explained, during the feedback workshop (B.FW.001), that he wanted representatives of the process areas to report poor progress, and that he wasn't interested in good progress.

Table 7.3.1 and Figure 7.3.1 also indicate that poor progress accounts for a substantial percentage of references to progress. For Project B, poor progress accounts for about 48% of the reports of progress of work. For Project C, poor progress accounts for about 45%. Taking a project-wide perspective, one may speculate that Project B as a whole is progressing worse than planned for almost 48% of the duration of the project, and Project C is also progressing worse than planned for about 45% of the duration of the project. This does not imply, however, that the project will take 48% longer duration or consume 48% more resource. (Indeed, chapter five indicates that the projects do not progress for 48% longer duration or consume 48% more resource.) Chapter five indicates, however, that there was an increase in resource, and an increase in effort (through overtime, shift-work etc.) and that some of the work was not completed when the product was delivered. The increase in capability and the reduction in workload indicates management taking action to respond to the poor progress of the project.

Given that status meetings tend to occur weekly, it would seem more reasonable to presume that representatives of process areas would tend to report on the weekly progress of their process area. Consequently, a reference to poor progress might indicate that a particular process area progressed poorly for (up to) a week of the project. So for Project B, the counts given in Table 7.3.1 suggest that particular process areas progressed poorly for 32 weeks of the project. (In this context, the term 'week' means a calendar week of effort at the level of process area).

Despite the similarities shown in Figure 7.3.1, there may be an underlying problem in comparing the two classifications because Project C conducts design/code/test status meetings, whereas Project B conducts project status meetings. This means that, for

Project C, references to poor progress may be at the level of the individual rather than the process area. Appendix B5 provides information on the breakdown of the initial types, including how the individual items of evidence were mapped to these initial types and then to the common types.

7.4 Process areas reporting the progress of work

Table 7.4.2 breaks down the types of progress of work per process area per project. Figure 7.4.1 provides a visual representation of Table 7.4.2, indicating the associations between types of progress and process areas.

Table 7.4.1 Significant values in Table 7.4.2

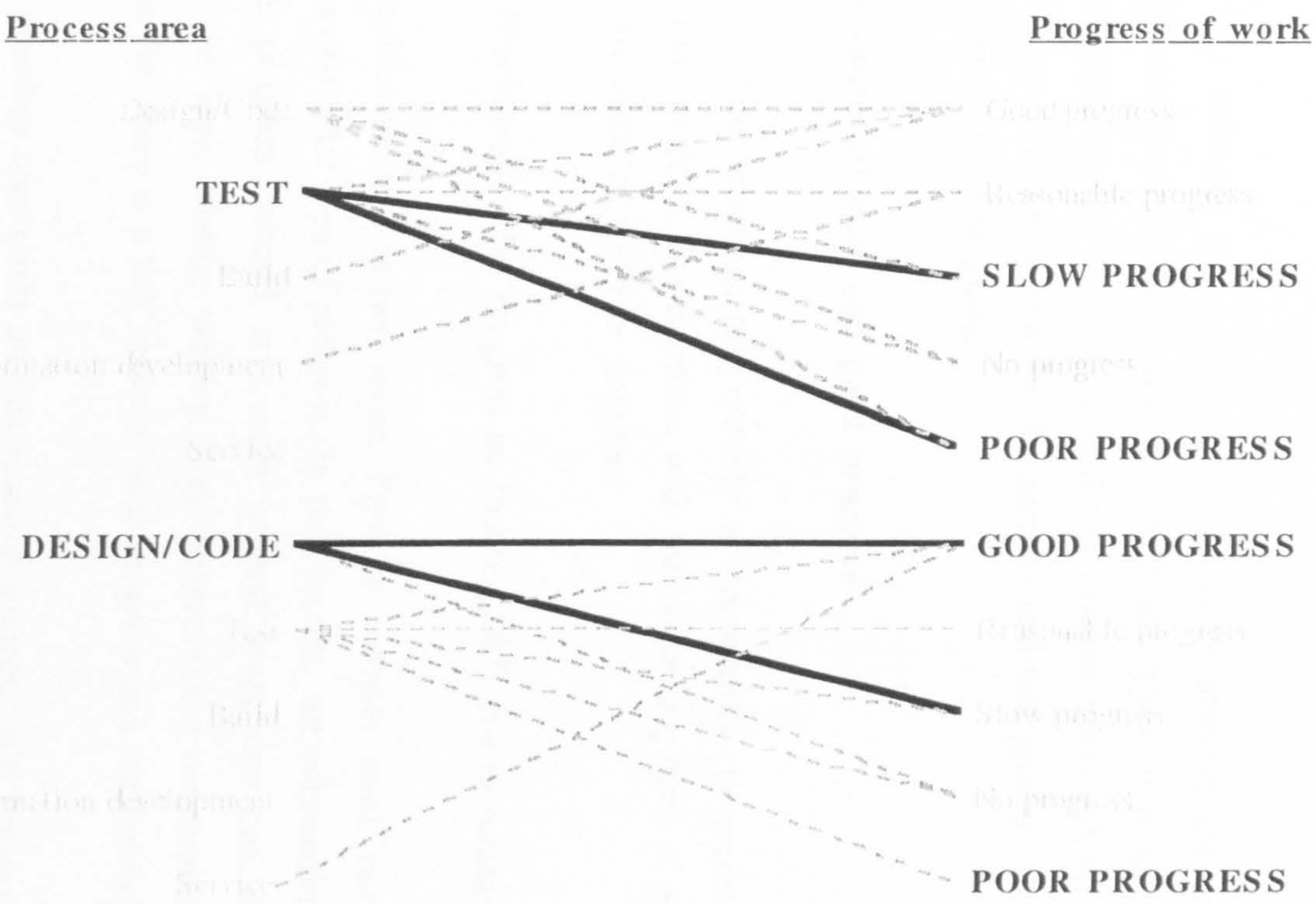
Project	Groups of cells	Number of cells	Total count	Mean	P≤0.05	P≤0.01	P≤0.001
B	'Internal' cells	20	51	2.55	9	10	11
B	Row totals	5	51	10.2	>19	>19	>19
B	Column totals	4	51	12.75	18	20	22
B	Poor progress	15	51	3.4	10	11	13
C	'Internal' cells	20	71	3.55	11	12	14
C	Row totals	5	71	14.2	>24	>24	>24
C	Column Totals	4	71	17.75	23	26	28
C	Poor progress	15	71	4.73	12	14	16

Table 7.4.1 presents the probability thresholds at which values in Table 7.4.2 become significant. Probabilities were calculated to a 95% confidence interval. A full explanation of how the probabilities were calculated is provided in Appendix A0. The Number of cells and Total count recorded in Table 7.4.1 is smaller than the actual number of cells and actual total count in Table 7.4.2 because the Unknown category has been removed from the calculations.

Table 7.4.2 Types of progress per process area

	Design/ Code		Test		Build		Information Development		Service		Unknown		Total	
Type of progress	**B**	**C**	**B**	**C**	**B**	**C**	**B**	**C**	**B**	**C**	**B**	**C**	**B**	**C**
Good progress	1	16	8	6	1		0			1	1	1	11	24
Reasonable progress	0	0	7	2	0		1			0	0	0	8	2
Slow progress	1	17	12	4	0		0			0	6	2	19	23
No progress	3	8	8	10	0		0			0	2	4	13	22
Unknown													16	28
Total	5	41	35	22	1		1			1	9	7	51	71
Poor progress	4	25	20	14	0		0			0	8	6	32	45

Figure 7.4.1 Types of progress per process area



In Figure 7.4.1, thick, black solid lines represent extremely significant associations (i.e. P≤0.001). Solid black lines represent very significant associations (i.e. P≤0.01). Broken black lines represent significant associations (i.e. P≤0.05). Grey, broken lines represent non-significant associations. Names of process areas that are emboldened and capitalised represent those process areas with an extremely significant number of references. Names of process areas that are emboldened represent those process areas with a very significant number of references. Names of process areas that are in normal text represent those process areas with a significant number of references. Names of process areas that are in grey text are not significant.

Figure 7.4.1 indicates that there is little similarity between the two projects, in terms of the significant process areas and their significant types of progress. Note, however, that with both projects the design/code and test process areas exhibit the most variability in their progress. Also, note that the design/code and test process areas are the only process areas to report poor progress.

7.5 The causes of poor and good progress

Minutes of status meetings also provide an opportunity to establish causal links between events in the projects. For example, with the following item of evidence it is possible to identify the cause of the poor progress:

> "Performance: No progress on [feature F07] due to outstanding sev 1 problem."

This provides insights at two levels. First, at a lower level, one can identify what has caused a particular type of progress. In this instant, an outstanding severity one problem prevents progress. Second, at a more general level, one can use these statements to support the logic relating waiting, poor progress and outstanding work. This example indicates that there is a link between poor progress and outstanding work.

Due to practical constraints on this research project, only Project B has been analysed in this way.

Table 7.5.1 Factors contributing to poor progress on Project B

Factor	Type of progress		
	No progress	Slow progress	Poor progress
Absence	1	0	1
Defect/Fix	4	6	10
Units of code for Build	1	0	1
Prep of Skills Transfer	1	1	2
Sickness	1	0	1
System reliability problems	2	4	6
Total	10	11	21

Table 7.5.1 breaks down the causes of poor progress identified for Project B. From 32 references to poor progress (see, for example, Table 7.4.2), it has been possible to identify the cause of poor progress for 21 references. It is clear from the table that the most salient cause of poor progress is Defects/Fixes, with System reliability problems also salient.

Table 7.5.2 Factors contributing to good progress on Project B

Factor	Count
Defects/Fixes	2
Developer started mission pay	1

Table 7.5.2 breaks down the causes of good progress identified for Project B. From 11 references to good progress (see, for example, Table 7.4.2), it has been possible to identify the cause of poor progress for three references. It is clear from the table that the most salient cause of good progress is Defects/Fixes. This is consistent with Table 7.5.1 *viz.* defects without fixes are preventing progress, whilst fixes to defects enable progress. 'Mission pay' is a method for improving progress by increasing effort on the project (though not necessarily increasing productivity[4]).

[4] Unpaid overtime has a subtle effect of appearing to increase productivity because the increased effort is not recorded but the increased output is.

7.6 Summary

This chapter has presented and discussed the analysis of progress evidence for Projects B and C. A number of insights have emerged from this analysis:

1. The reporting of progress does not appear to be only a function of progress itself.
   - The presence of major milestones (internal or external milestones) may influence the reporting of progress.
   - Similarly, the increase in project activity may influence the reporting of progress.
   - It might be that progress is not reported in the early stages of the projects because it is difficult to assess progress during these stages.
   - Managers may only report exceptions.
   - While managers may report all progress, it may be that only poor progress is recorded in the minutes of the status meetings.
2. For the process areas of the project, reports of poor progress are more prevalent during the end (the test phase) of the project than during the middle (the design/code phase) of the project.
3. The two projects are similar in their reporting of progress:
   - There is more reporting of poor progress (and this reporting is substantial) than good or reasonable progress.
   - There is some suggestion for a bi-modal distribution of reporting good or poor progress but not reporting reasonable progress.
   - There is some suggestion that reports of any kind of progress tend to occur during periods of poor progress.
   - Only the design/code and test process areas report poor progress.
4. While other process areas report progress, the design/code and test process areas make the most reports of progress (whether poor, reasonable or good).
5. Defect/Fixes is the only factor that clearly affects progress, with the absence of a fix contributing to poor progress and the presence of a fix contributing to good progress.

Chapter 8 Outstanding work

8.1 Introduction

This chapter presents and discusses the analysis of the outstanding work evidence for Projects B and C. The evidence is used in three ways. First, to provide insights into the nature of process areas within a project. Second, to provide evidence relating to the model of capability (recall from chapter four that the presence of outstanding work is assumed to be caused by poor progress, and is also assumed to result in waiting and a threat to subsequent capability). Third, to provide evidence to test Bradac *et al.*'s conjecture that waiting is more prevalent during the end of the project than during the middle of the project.

The specific questions investigated in this chapter are:

- What is the frequency of references to outstanding work?
- What is the prevalence of outstanding work over the duration of the project? This is a complementary test of Bradac *et al.*'s conjecture that waiting is more prevalent during the end of the project than during the middle of the project.
- What are the different types of outstanding work, and what are their frequencies?
- What is the breakdown of the types of outstanding work against process areas of the project?

Details on the methods used to collect, organise and analyse the outstanding work evidence are provided in chapter three.

8.2 The frequency and prevalence of outstanding work

The frequency of outstanding work

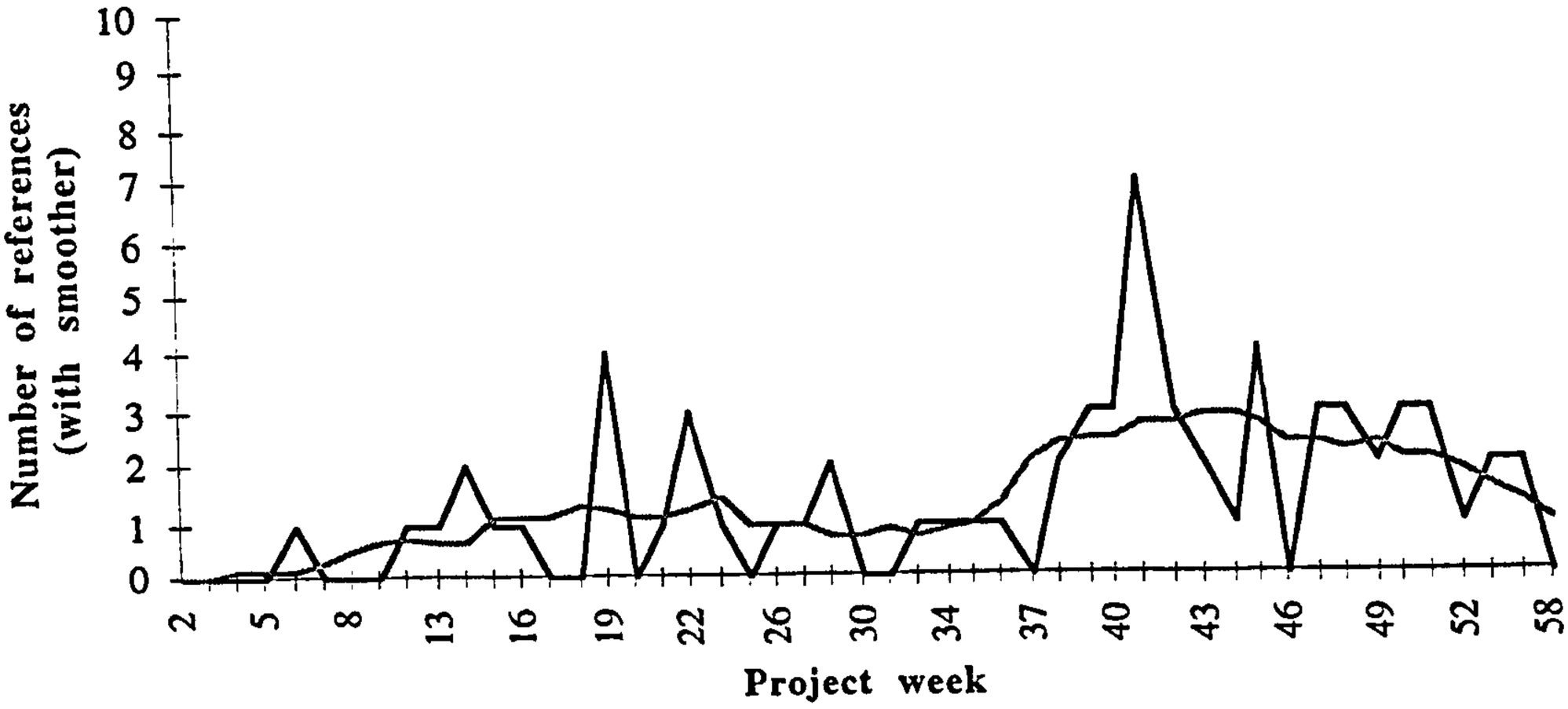


Figure 8.2.1 References to outstanding work for Project B

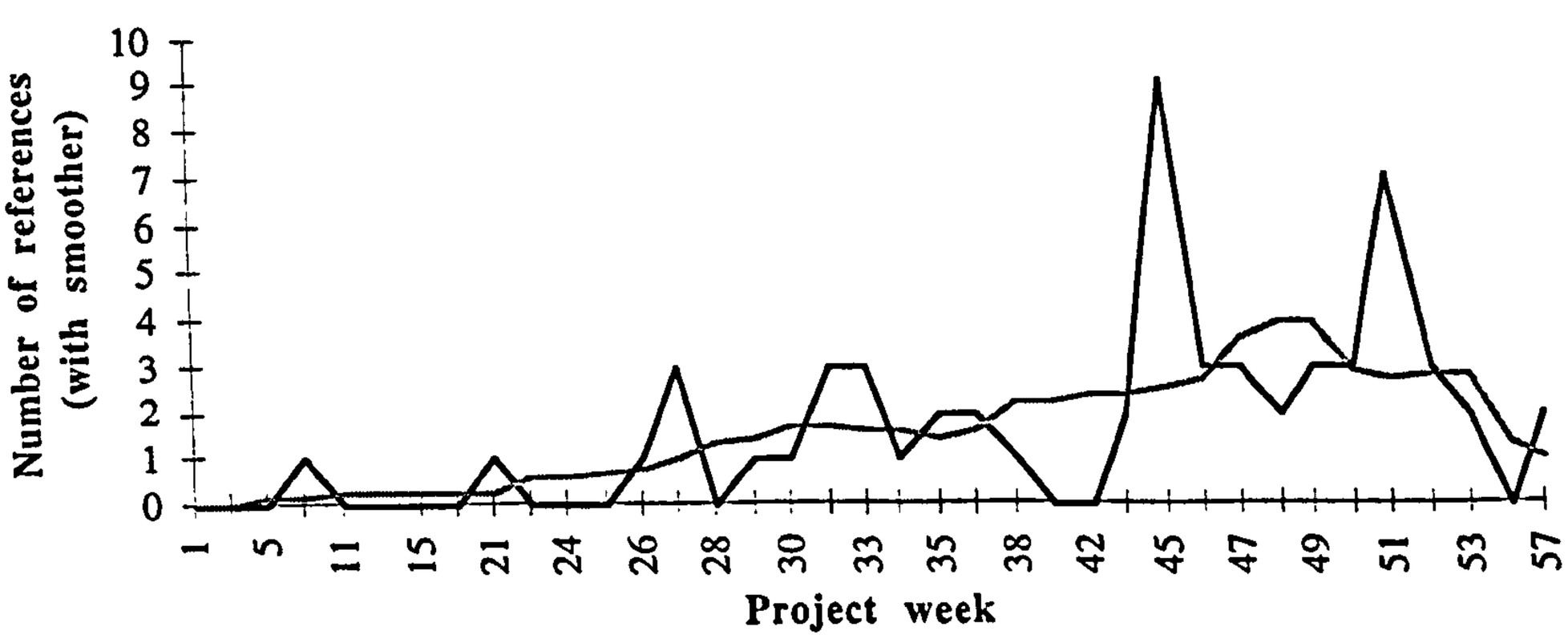


Figure 8.2.2 References to outstanding work for Project C

Table 8.2.1 Summary statistics for the frequency of outstanding work

Project	Weeks	Mean	Median	Mode	Min	Max	Range	Total
B	49	1.4	1	0	0	7	7	68
C	37	1.6	1	0	0	9	9	59

Figures 8.2.1 and 8.2.2 present lineplots of the frequency of references to outstanding work for Projects B and C. The plots include a smoother, calculated as a moving average over a range of nine datapoints. (Precise values are presented in Appendix B6.) Only the

weeks for which a status meeting occurred are included in the plots. Table 8.2.1 presents summary statistics to accompany Figures 8.2.1 and 8.2.2.

The table and the figures indicate that there is a good degree of similarity between the two projects, with the median and mode identical for the two projects and the mean slightly larger for Project C. The maximum number of references to outstanding work is greater for Project C (nine compared to seven), but the total number of references is greater for Project B (68 compared to 59). This is probably partly due to a greater number of status meetings for Project B (49 compared to 37). For both projects it is clear that the number of references to outstanding work fluctuate over the weeks of the projects. Nevertheless, the presence of outstanding work throughout the duration of the project suggests frequent poor progress (*cf.* chapter seven).

For Project B, there is a noticeable spike in week 41, the week the decision was made to remain committed to the original plan (see chapter five for more information). The sudden incline of the smoother at week 36 occurs during the period that the design/code phase actually finished. In Figure 5.3.3, from week 36 there is a noticeable increase in the indicators of project activity. Chapter five has suggested that the indicators of project activity may also be examples of the tactics of management, and as such they suggest that the project's management are responding to perceived imbalances between workload and capability.

For Project C, there are noticeable spikes at week 45 and week 51. There are no obvious events that occurred during week 45, but in week 51 the third re-plan occurred. The 'bump' in the smoother between weeks 46 and 51 is 'artificially' caused by the spikes at weeks 45 and 51. Unlike Project B, a pattern of project activity relating to the frequency of outstanding work is not apparent for Project C.

The prevalence of outstanding work

The previous two chapters have tested the prevalence of reporting waiting and the poor progress of work during the middle and end of the projects. A similar test can be conducted for the prevalence of outstanding work. Explicitly:

$H3_{Exp}$ For the process areas of the project, outstanding work is more prevalent during the end of the project than during the middle of the project

Table 8.2.2 Results of the Mann Whitney *U* tests of hypothesis $H3_{Exp}$

Project	*p*	∝	N
B	0·0197	0·05	46
C	0·0004	0·05	33

Table 8.2.3 Summary statistics for the prevalence of outstanding work

Project	Project stage	Weeks	Median	Min	Max	Range
B	Middle	17	1	0	4	4
	End	29	2	0	7	7
C	Middle	13	0	0	3	3
	End	20	2	0	9	9

Table 8.2.2 presents the results of the Mann Whitney *U* tests of hypothesis $H3_{Exp}$ for the two projects. Tied values were included in the test. Details on the test are provided in Appendix B6. Table 8.2.3 provides summary statistics. Both tests reject the null hypothesis and retain the experimental hypothesis, at an alpha-level of 0·05. Thus, the reporting of outstanding work is more prevalent during the end of the project than during the middle of the project. Based on the logic presented in chapter four, that outstanding work leads to waiting, the two tests of hypothesis $H3_{Exp}$ support Bradac *et al.*'s conjecture that waiting is more prevalent during the end of the project than during the middle of the project. The Mann Whitney *U* tests also suggest that imbalances between capability and workload are more prevalent during the end of the project than during the middle of the project.

8.3 Types of outstanding work, and their frequencies

Table 8.3.1 Types of outstanding work

Type of outstanding work	Project B Count	Project B %	Project C %	Project C Count
Design/Code	0	0	28.8	17
Decision	9	13.2	0	0
Defects/Fixes	37	54.4	18.6	11
Tests	7	10.3	17.0	10
Problem	8	11.8	13.6	8
Publications	2	2.9	10.2	6
Other	4	5.9	8.4	5
Unknown	1	1.5	3.4	2
Total	68	100	100	59

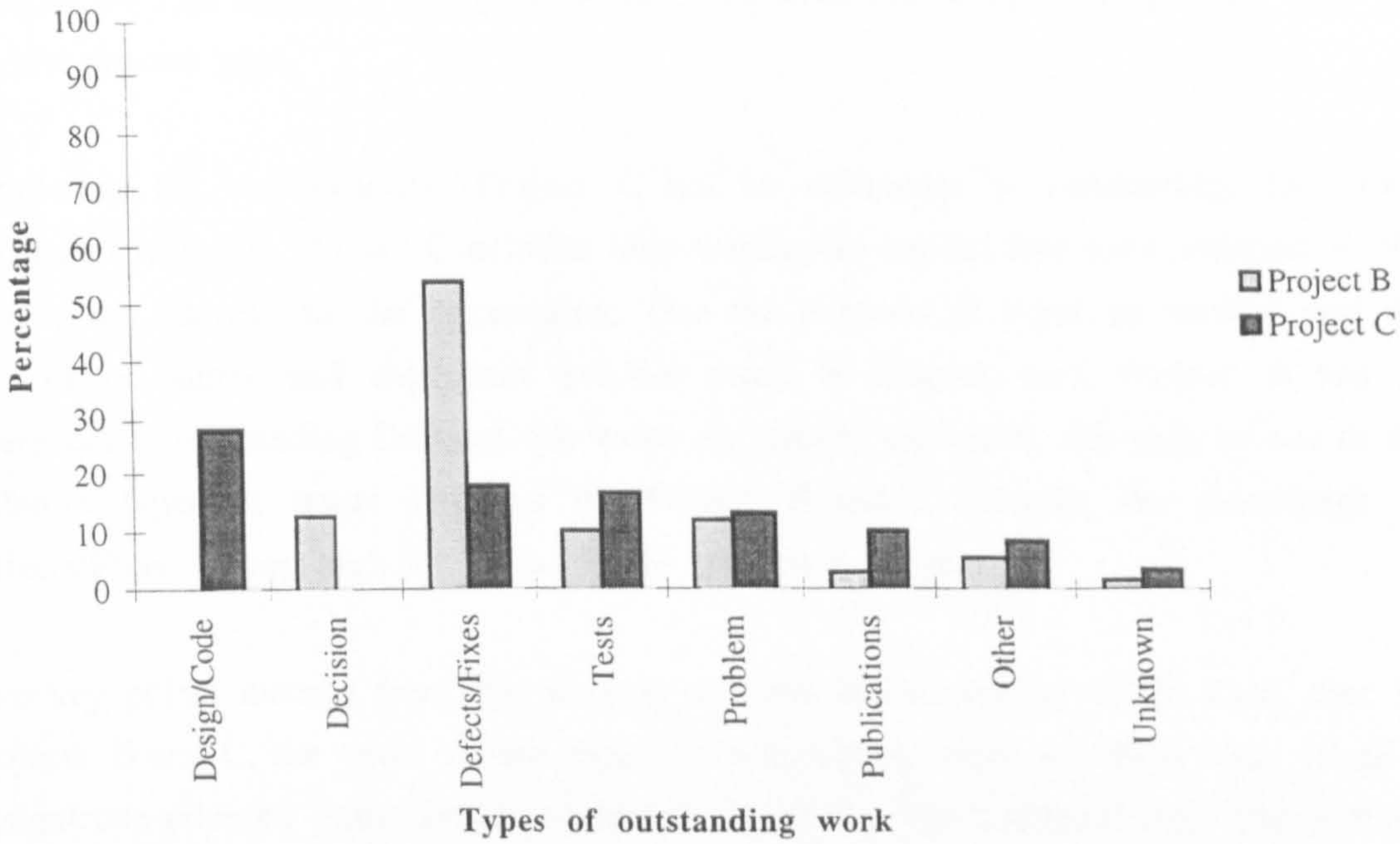


Figure 8.3.1 Types of outstanding work

Table 8.3.1 and Figure 8.3.1 break down the types of outstanding work for the two projects. For Project B, Defect/Fixes clearly dominate. In addition to Defects/Fixes, Decisions, Problems and Tests are also salient types of outstanding work. Problems may be yet-to-be identified Defects.

The percentage of outstanding Defects/Fixes suggests a strong relationship between the test process area, who typically identify the defects, and the design/code process area, who typically provide the fixes. The percentage of outstanding Tests may also be an indicator of this relationship. If Problems are yet-to-be identified Defects, then the presence of outstanding Problems would further indicate the strong relationship between

the design/code process area and the test process area. This relationship is also supported by the analysis of the waiting evidence and the progress of work evidence (see chapters six and seven respectively). Overall, this suggests that imbalances in the design/code process area results in outstanding work; work that the test process area is waiting on, and which threaten the capability of the test process area.

For Project C, Design/Code, Defects/Fixes, Tests, Problems and Publications are all salient types of outstanding work. The relatively high percentage of outstanding Design/Code work may be explained by the fact that the minutes of the design/code/test status meetings were used in the analysis of Project C, whereas the minutes of project status meetings were used in the analysis of Project B. Like Project B, the percentages of Defect/Fixes, Tests and Problems (together, perhaps, with the percentages of Design/Code) all suggest a strong relationship between the design/code process area and the test process area.

Comparing the two projects, Project C has no references to outstanding Decisions. Consistent with this, Project C exhibits little waiting on entities that were external to the project but internal to the organisation (see the analysis of types of waiting and the analysis of source and dependent process areas in chapter six). Project B has no references to outstanding Design/Code work. As already explained, this may be due to the different types of status meetings for Project B and C. Finally, the percentage of Defects/Fixes is very high for Project B, in contrast to Project C.

Two key points emerge from the analysis of types of outstanding work. First, that for Projects B and C, the most salient types of outstanding work are those that relate to design/code-oriented issues and test-oriented issues (i.e. the Defects/Fixes, Design/Code, Tests and Problems types). This suggests the prominence of the design/code and test process areas within these two software development projects. Second, that there may be a strong relationship (dependency) between the test process area and the design/code process area.

8.4 Process areas reporting outstanding work

Table 8.4.2 breaks down outstanding work per process area for the two projects. Cells with no value indicate situations where the process area or type of outstanding work was not referenced in that project, but was referenced in the other project. Figure 8.4.1 provides a visual representation of Table 8.4.2. For Project B, the Defect Screen Team was created at the beginning of the project to manage the allocation of defects to developers. The Defect Screen Team changed from weekly meetings to daily meetings in

week 38, two weeks after the completion of the design/code phase (see chapter five for more information). Typically, a Defect Screen Team would be formed later in a project at this organisation (perhaps around the time that the design/code phase completes and the test phase commences). It is not clear whether a Defect Screen Team actually existed for Project C. Outstanding work by the Defect Screen Team can be related to the outstanding Defects/Fixes and outstanding Tests, because the Defect Screen Team decide the priority of the defect and allocate that defect to a fixer. While the Defect Screen Team may help to manage defect fixing, it also acts as a potential bottleneck in the defect fixing process.

Table 8.4.1 Significant values in Table 8.4.2

Project	Groups of cells	Number of cells	Total count	Mean	P≤0.05	P≤0.01	P≤0.001
B	'Internal' cells	36	56	1.6	7	8	9
B	Row totals	6	56	9.3	17	19	21
B	Column totals	6	56	9.3	17	19	21
C	'Internal' cells	36	48	1.3	7	8	9
C	Row totals	6	48	8	16	17	≥18
C	Column Totals	6	48	8	16	17	19

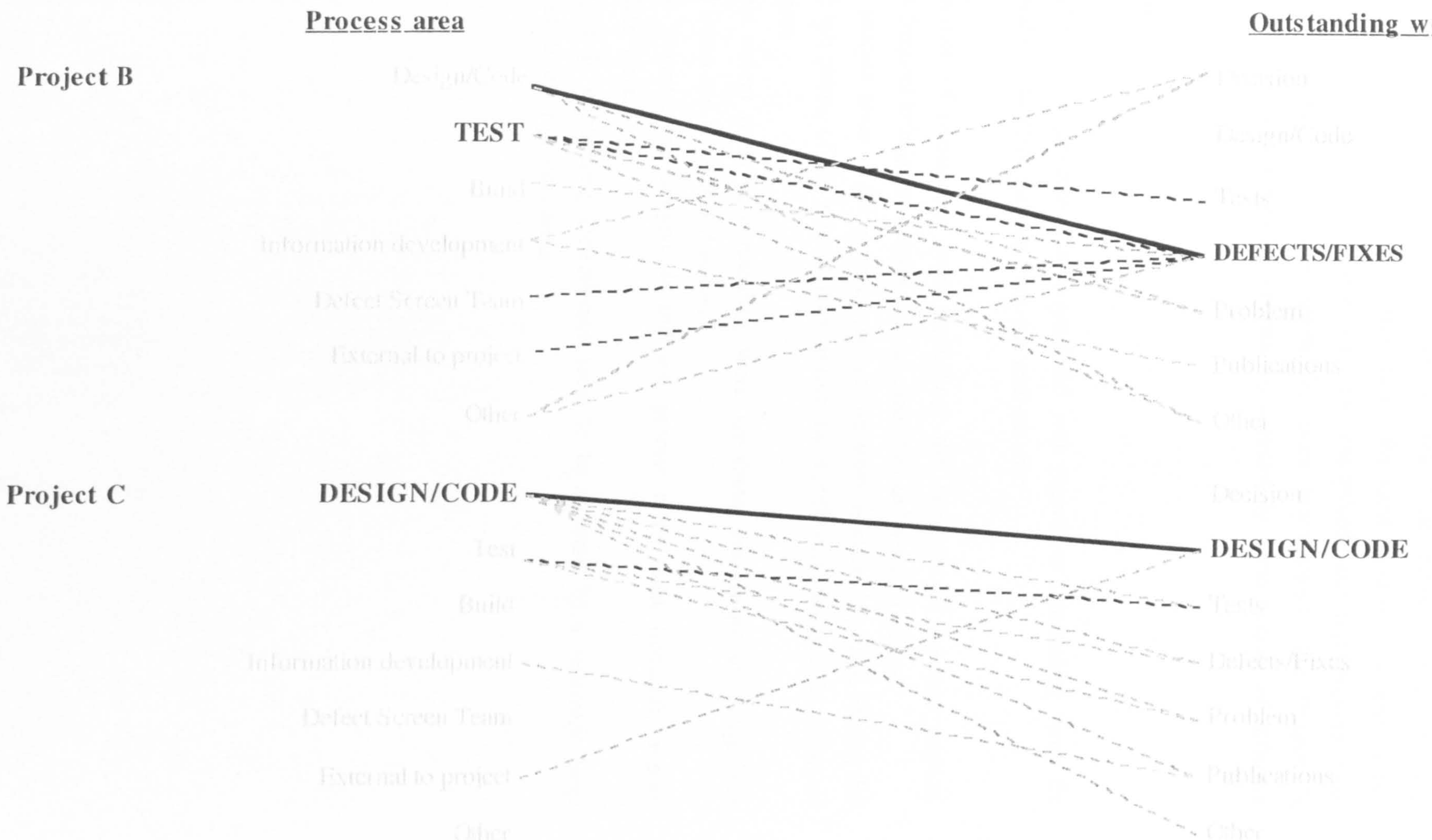
Table 8.4.1 presents the probability thresholds at which values in Table 8.4.2 become significant. Probabilities were calculated to a 95% confidence interval. A full explanation of how the probabilities were calculated is provided in Appendix A0. The Number of cells and Total count recorded in Table 8.4.1 is smaller than the actual number of cells and actual total count in Table 8.4.2 because the Unknown category has been removed from the calculations.

For Figure 8.4.2, thick, black solid lines represent extremely significant associations (i.e. P≤0.001). Solid black lines represent very significant associations (i.e. P≤0.01). Broken black lines represent significant associations (i.e. P≤0.05). Grey, broken lines represent non-significant associations. Names of process areas that are emboldened and capitalised represent those process areas with an extremely significant number of references. Names of process areas that are emboldened represent those process areas with a very significant number of references. Names of process areas that are in normal text represent those process areas with a significant number of references. Names of process areas that are in grey text are not significant.

Table 8.4.2 Relationships between types of outstanding work and process areas

Type of outstanding work	Design / Code		Test		Build		Information development		Defect Screen Team		External to project		Other		Unknown		Total	
	B	C	B	C	B	C	B	C	B	C	B	C	B	C	B	C	B	C
Decision	0		0		0		4		0		0		5		0		9	
Design/Code		16		0				0				1			0			17
Defects/Fixes	12	6	7	3	1		0	0	8		8	0	1		0	2	37	11
Tests	0	3	7	7	0		0	0	0		0	0	0		0		7	10
Problem	1	5	6	2	0		0	0	0		0	0	1		0	1	8	8
Publications	0	4	0	0	0		2	1	0		0	0	0		0	1	2	6
Other	2	3	1	0	0		0	0	0		0	0	0		1	2	4	5
Unknown	0	1	1	0	0		0	0	0		0	0	0		0	1	1	2
Total	15	38	22	12	1		6	1	8		8	1	7		1	7	68	59

Figure 8.4.1 Process areas and their relationships with outstanding work



It is clear from Figure 8.4.2 that there is little similarity between the two projects in terms of the process areas reporting outstanding work. Note, however, that for both projects the design/code and test process areas report the most number of different types of outstanding work. This is consistent with the design/code and test process areas reporting poor progress (see chapter seven for more information). For Project B, outstanding Defect/Fixes seem to be reported by a relatively large number of different process areas.

8.5 Summary

A number of research questions were investigated with regards to outstanding work for Projects B and C. The main insights from this analysis are:

1. The presence of outstanding work throughout the durations of the two projects suggests frequent imbalances between actual capability and actual workload.
2. For the process areas of the project, outstanding work is more prevalent during the end (the test phase) of the project than during the middle (the design/code phase) of the project.
3. The most salient types of outstanding work are those that relate to design/code oriented issues and test oriented issues.
4. For both projects, the design/code and test process areas most frequently report outstanding work.
5. For Project B, outstanding Defect/Fixes seem to be reported by a relatively large number of different process areas.

Chapter 9 Integrating the analyses

9.1 Introduction

A considerable amount of evidence and complex analyses have been presented in the preceding five chapters. This chapter completes this analyses. Specifically it:

1. Provides a summary of the various insights gained into the two projects.
2. Relates these insights to:
   - Bradac *et al.*'s research
   - The model of capability
   - The model of software project schedule behaviour.
3. Seeks support for these insights from previous research.
4. Generalises these insights to other projects.

9.2 A summary of the various insights into Projects B and C

Table 9.2.1 summarises the insights gained into the behaviours of Project B and Project C. It is clear from the table that there are many insights common to the two projects, although there are also some notable differences, particularly with regards to the socio-technical contexts of the two projects.

The Project Leaders of Project B and Project C both consider their projects to be successful. Unsurprisingly, their assessment of the success of their respective projects is based on a combined set of technical and business criteria (*cf.* [36]). An assessment of the success of Project B and Project C based on the integrated model of project schedule behaviour and capability would focus on only two criteria:

1. An assessment of whether the actual elapsed time to product delivery matched the originally planned elapsed time to product delivery.

2. An assessment of whether the actual elapsed time for the entire project matched the originally planned elapsed time for the entire project[5].

For both criteria, Project B is successful, because it delivered the product when originally planned and completed the project when planned (see insights #1.1 and #1.2 in Table 9.2.1). For both criteria, Project C is unsuccessful because it delivered the product later than originally planned and completed the project later than originally planned (again, see insights #1.1 and #1.2 in Table 9.2.1). The integrated model of schedule behaviour and capability should explain why Project B was successful and why Project C was unsuccessful. Surprisingly, despite the differences in the success of the two projects, the behaviour of the two projects appears to be very similar, with the exception of their socio-technical contexts. This would suggest that, based on the integrated model, the socio-technical contexts principally account for the schedule success of Project B and the schedule 'failure' of Project C. (There may, of course, be factors that are not represented by the integrated model that account for the schedule success of Project B and the schedule 'failure' of Project C.)

[5] A more accurate assessment would assess the *degree* to which the actual elapsed time to product delivery matched the planned elapsed time to product delivery, and the *degree* to which the actual elapsed time for the entire project matched the planned elapsed time for the entire project.

Table 9.2.1 A summary of the insights into Project B and Project C

#	Insight	Project
1	**Project-level schedule**	
1.1	Actual elapsed time for the entire project was:	
	• As originally planned	B
	• Longer than originally planned	C
1.2	Actual elapsed time to deliver the product was:	
	• As originally planned	B
	• Longer than originally planned	C
2	**Phase-level schedule (the actual progress of phases)**	
2.1	Phases were originally planned to occur sequentially but actually occurred concurrently	B & C
2.2	The plan phase completed later than planned	B & C
2.3	The design/code phase started when planned	B & C
2.4	The design/code phase completed later than planned	B & C
2.5	The test phase started when planned	B & C
2.6	The test phase completed later than planned	B & C
2.7	The design/code phase and test phase progressed concurrently	B & C
2.8	Overall, the project's phases did not actually progress according to the original plan	B & C
3	**Actual workload**	
3.1	The project experienced significant explicit increases in workload (i.e. new features or design changes)	B & C
3.2	The project experienced significant implicit increases in workload (i.e. design rework)	B & C
3.3	The project experienced implicit decreases in workload (see the tactics of management below)	B & C
3.4	Overall, the project's actual workload was not as originally planned, and was actually more than planned	B & C
4	**Actual capability**	
4.1	The project experienced overt increases in capability	B & C
4.2	The project experienced covert increases in capability (see the tactics of management below)	B & C

Table 9.2.1 A summary of the insights into Project B and Project C (*continued*)

#	Insight	Project
5	**Waiting, progress of work and outstanding work**	
5.1	The project recorded more reports of poor progress than of good progress	B & C
5.2	The project recorded very few reports of reasonable progress	B & C
5.3	Only the design/code and test process areas reported poor progress	B & C
5.4	The design/code and test process areas reported the most progress	B & C
5.5	Waiting, poor progress and outstanding work were more prevalent during the end (the test phase) of the project than during the middle (the design/code phase) of the project	B & C
5.6	Reports of waiting, poor progress and outstanding work related principally to the design/code and test process areas	
	• They were the most frequently referenced process areas within each set of evidence	B & C
	• They were the most frequently referenced process areas across all sets of evidence	B & C
5.7	In terms of waiting and outstanding work, the most references were to:	
	• The Defect/Fixes type of work	B
	• The Design/Code type of work	C
	• The Decisions type of work	B
5.8	The principal causes of poor progress and good progress were Defects/Fixes. (This was only investigated for Project B.)	B
5.9	Reports of waiting, progress of work and outstanding work referred to external process areas (*cf.* #6.11 and #7.5 below)	B
6	**Tactics of management**	
6.1	The project re-planned the internal schedule in order to maintain external commitments	B & C
6.2	The project found covert ways to increase its capability	B & C
6.3	The project found covert ways to reduce its workload	B & C
6.4	A number of internal re-plans occurred	B & C
6.5	One or more external re-plans occurred (*cf.* #1.1 and #1.2)	C
6.6	Re-plans (whether internal or external) only started to occur after the design/code phase was planned to complete.	B & C
6.7	The internal re-plans focused on manipulating the phases of the project, rather than the workload or capability	B & C
6.8	The external re-plans focused on manipulating the phases of the project, rather than the workload or capability	C
6.9	The re-planning processes were considerably shorter than the original planning process	B & C
6.10	The expression of the internal re-plans did not appear to be communicated formally	B & C
6.11	Re-plans responded to:	
	• Changes in expected and planned events	B & C
	• Changes in unexpected and unplanned events	C
	• Internal events	B & C
	• External events (*cf.* #7.5 below)	C

Table 9.2.1 A summary of the insights into Project B and Project C (*continued*)

#	Insight	Project
7	**Socio-technical context**	
7.1	The project was end-date driven, due to:	B & C
	• Marketing considerations	B
	• Funding constraints	C
7.2	The strategic value of the product to the organisation was:	
	• Higher and long-term	B
	• Lower and short-term	C
7.3	The product was:	
	• A legacy, middle-ware, mission-critical mainframe software system	B
	• A legacy, middle-ware workstation and desktop software system	C
7.4	The purpose of the project was:	
	• Enhancing the existing product	B
	• Porting the product to a new platform	C
7.5	For the project, the presence of:	
	• External dependencies was significant	B
	• External dependencies was not significant	C
7.6	*Project* status meetings occurred?	
	• Yes	B
	• No (but design/code/test status meetings)	C
7.7	The project was managed with a multi-functional management team	B & C
7.8	The role(s) of the Project Leader were:	
	• The Project Leader's sole role was Project Leader	B
	• The Project Leader was also Development Manager and Support Manager	C
7.9	The project had:	
	• Distinct development and support teams	B
	• Development and support work allocated across the development and support teams	C

A closer inspection of Table 9.2.1 indicates a substantial number of insights referring to design/code-related and test-related issues. Both the design/code and test phases completed later than planned and progress concurrently with each other (see insights #2.4, #2.6 and #2.7). Only the design/code and test process areas reported poor progress (see insight #5.3). Reports of waiting, poor progress and outstanding work related principally to the design/code and test process areas (see insight #5.6). The most references, within the waiting and outstanding work evidence, were to design/code and test types of work (see insight # 5.7). The principle causes of poor progress and good progress were Defects/Fixes (see insight #5.8), a type of work involving both the design/code and test process areas. Re-plans did not start to occur until the completion of the design/code phase and the start of the test phase (see insight #6.6).

Also, external events were referred to by a number of insights, but there is more complexity with these references. For Project B only, reports of waiting, progress of work and outstanding work all refer to external process areas (see insight #5.9). For Project C only, re-plans respond to changes in external events (see insight #6.11). For Project C, the presence of external dependencies are not significant, but external dependencies are significant for Project B (see insight #7.5). This suggests that from the beginning of the project, and throughout the project, Project B is aware of and must manage relationships with external process areas (e.g. other projects within the corporation). Project B is also able to *manage* these relationships effectively; they are expected and effectively planned for. By contrast, from the beginning of the project, and throughout much of the project, Project C does not have external dependencies. The external re-plan is in response to an unexpected and unplanned for external event i.e. the introduction of new year-2000 requirements.

9.3 Relating the insights to Bradac *et al.*'s research

The prevalence of waiting, poor progress and outstanding work during the end of the project (the test phase) rather than during the middle of the project (the design/code phase) individually and collectively provide evidence to support the conjecture of Bradac *et al.* that waiting is more prevalent during the end of the project than during the middle of the project. The evidence and analyses presented in this thesis complements, rather than replicates, Bradac *et al.*'s work because this evidence relates to process areas, in contrast to Bradac *et al.*'s investigation of an individual designer.

In chapter two, the point was made that two subsequent studies (i.e. [5, 34]) have *assumed* Bradac *et al.*'s conjecture to be valid. The evidence and analysis presented here not only strengthens the validity of Bradac *et al.*'s conjecture, but also strengthens the

validity of the subsequent studies, because this study has tested the assumptions made by those two studies.

The analyses presented in this thesis also indicates, however, that the dominant type of waiting is 'Waiting on software' (see chapter six for more information). This contrasts with the work of Bradac *et al.*, who observed that the dominant type of waiting was 'Waiting on other'. Further investigation of the Waiting on software category has distinguished between waiting on code and waiting on fixes to defects in the code. Chapter six considered why there should be differences between the findings of Bradac *et al.* and the findings of this study. Broadly, this may be due to differences in the focus of the two studies, with Bradac *et al.* investigating an individual designer and this study investigating process areas. (Chapter three identifies a number of differences in the designs of the two studies, which might account for the difference in the two studies' findings.)

Chapter two identified two reasons for the unexpected use of time. The first reason, waiting, is based on Bradac *et al.*'s analysis. The second reason, implicit within Bradac *et al.*'s work, is that designers are unexpectedly reassigned to higher priority projects. In both Projects B and C, the reassignment of resource is apparent. For Project B, resource is retained by Project B rather than reassigned (when planned) to the succeeding project. In Project C, resource assigned to support is reassigned to new development. Once again, this provides complementary support to the work of Bradac *et al.* Unlike Bradac *et al.*'s work, however, it has not been possible to examine the relative frequencies of waiting and the reassignment of resource in this thesis.

Taking a broader perspective than just waiting, the analysis conducted as part of this investigation complements the studies of time usage presented and discussed in chapter two. This investigation provides insights into how process areas and projects actually use time. These insights are more abstract than the studies of how individual designers use their time.

9.4 Relating the insights to the model of capability

Given the fact that the waiting, poor progress and outstanding work evidence all have a similar prevalence (i.e. being more prevalent during the end of the project than during the middle of the project), this provides a further opportunity to explore the model of capability.

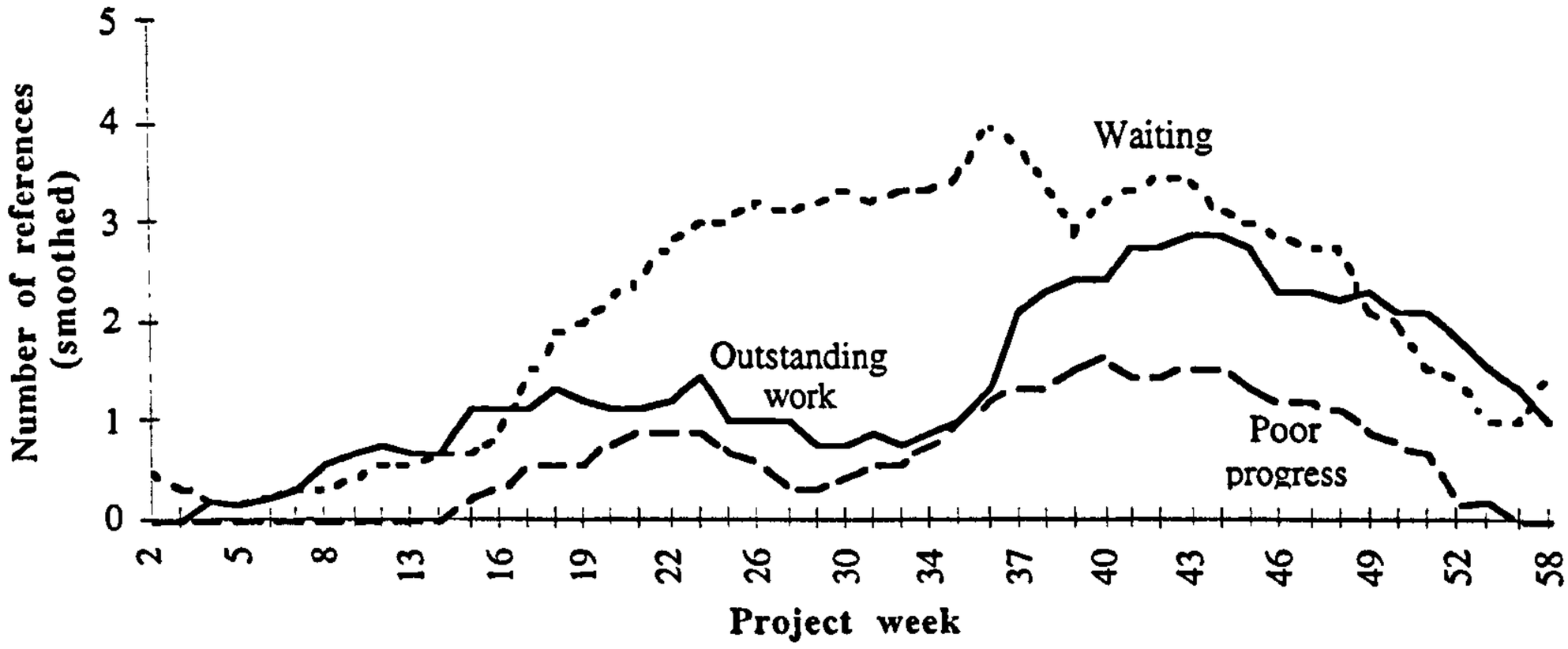


Figure 9.4.1 Smoothed frequency of the reporting evidence for Project B

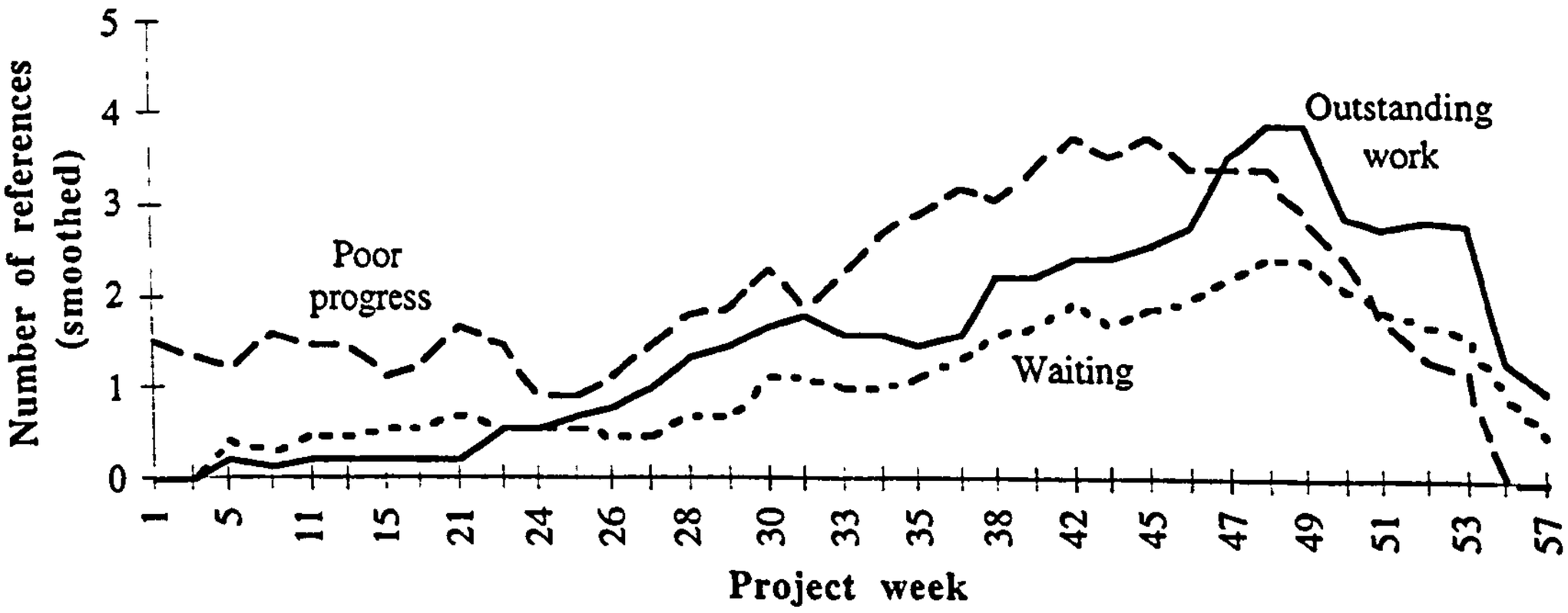


Figure 9.4.2 Smoothed frequency of the reporting evidence for Project C

Figures 9.4.1 and 9.4.2 provide a comparison of the smoothed frequencies of waiting, poor progress and outstanding work for the two projects. The smoothers are taken from figures presented in chapters six through eight, and are calculated as moving averages with a range of nine datapoints. Only the weeks where status meetings occur are included in the two figures. The weekly references are not cumulative.

For Project B, there is a similar pattern of frequencies between poor progress and outstanding work, with two 'waves' of reporting; one approximately between weeks 2 and 30 and one approximately between weeks 34 and 58. The similar pattern lends support to the logic that poor progress leads to outstanding work. A potential difficulty with this logic, however, is that some outstanding work is reported before poor progress is reported i.e. between weeks 3 and 14. This discrepancy might be explained by: the feedback relationship in the model; the tactics of management used to compensate for poor progress; and the possibility that the reporting of poor progress and outstanding work may not just be functions of the actual occurrence of poor progress and outstanding work.

The pattern of frequencies between outstanding work and waiting are not similar, primarily because of the 'bump' in the waiting evidence between weeks 18 and 39. In chapter five, Figure 5.3.1 shows that most of the increases in the number of design changes on Project B occurred between weeks 18 and 39. Also, there is a long period, between weeks 22 and 36, where there are no increases in design changes, and then in week 37 two further design changes are added to the project. The 'bump' in the frequency of waiting in Figure 9.4.1 might reflect a situation were the project is waiting on decisions regarding the acceptance of additional design changes. Chapter five also explains that at every status meeting for Project B the first item on the meeting agenda is to review the design changes. Consequently, it might be that the reporting of waiting on design changes bias the overall references to waiting. If one excludes the 'bump' in waiting then there is a similar pattern of frequencies between outstanding work and waiting, and this lends support to the logic that outstanding work leads to waiting.

Similarly, if one again excludes the 'bump' in the waiting evidence, then there is a similar pattern of frequencies between waiting and poor progress. This lends support to the logic that waiting threatens poor progress (via affecting capability). If one includes the 'bump' in the waiting evidence, the dissimilarities between the frequencies of waiting and poor progress (and outstanding work and waiting) might be explained by the possible influences of feedback; the tactics of management; and the fact that reports of waiting may not only be a function of waiting. There is also the issue that waiting *threatens* capability rather than always reducing capability. This is because time spent waiting in one activity may be effectively directed at another independent activity ([34]). Also, poor progress is defined as an imbalance between workload and capability. Consequently, poor progress might be due to an increase in workload rather than a reduction in capability. Chapter five shows clear increases in workload for Project B.

For Project C, there is a similar pattern of frequencies between poor progress and outstanding work, with some 'lag' between poor progress and outstanding work. (Compare

the fluctuation at week 30 for poor progress with week 38 for outstanding work; the peaks between weeks 42 and 45 for poor progress with weeks 48 and 49 for outstanding work; and the decline in outstanding work from weeks 49 through 57, which follows the decline in poor progress from weeks 48 through 54). There are also similar patterns of frequencies between outstanding work and waiting, and between waiting and poor progress.

For both projects, there is a general tendency for all three sets of evidence to increase, to plateau, and then to decline in the final weeks of the projects. This provides more detail to complement the results of the Mann Whitney U tests of the prevalence of waiting, poor progress and outstanding work.

Given the fact that feedback systems tend to behave counter-intuitively, and that a feedback loop is present in the model of capability, then the *obvious* patterns in the evidence might be surprising. The lack of counter-intuitive behaviour might be explained by the granularity of the evidence, caused by the frequency of the meeting minutes. For example, if one observes waiting, poor progress and outstanding work on a daily basis one might find more complex, and counter-intuitive, behaviour. The fact that status meetings occur weekly (and sometimes fortnightly) might simplify the relationships between waiting, poor progress and outstanding work because these three phenomena are all reported at the same time.

As noted in section 9.2, the frequency of references to the design/code and test process areas across all sets of evidence suggests that a validation of the model for only the design/code and test process areas might be effective. The frequency of references to the Defects/Fixes type of work across all sets of evidence suggests that a validation of the model using only the evidence referring to Defects/Fixes might also be effective. These two tests stand as opportunities for further research.

Overall, and excepting the caveat regarding feedback, the analysis conducted above lends additional support to the claim that the model of capability possesses descriptive and explanatory value.

9.5 Relating the insights to the model of software project schedule behaviour

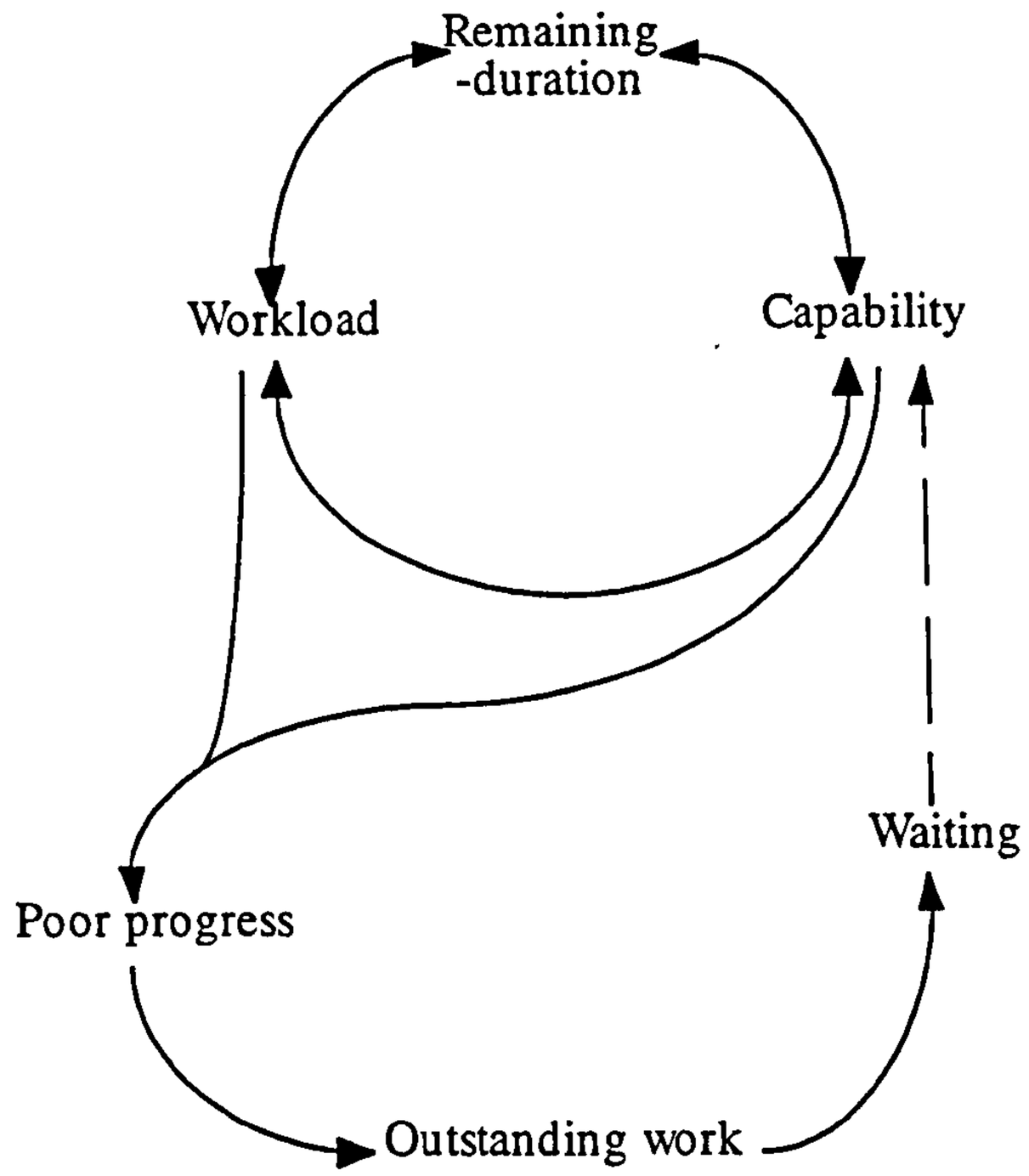


Figure 9.5.1 An integrated model of schedule behaviour and capability

Figure 9.5.1 re-presents the integrated model of schedule behaviour and capability. The model indicates how the behaviour of poor progress, outstanding work and waiting affect a project's remaining-duration:

1. The prevalence of poor progress towards the end of the project suggests that the imbalance between workload and capability is more common during the end of the project (but note that reports of progress do not appear to be only a function of progress itself). Evidence from chapter five shows that workload explicitly and implicitly increases during the middle (the design/code) and the end (the test phase) of the project, and evidence from chapter eight shows that workload implicitly increases through outstanding work (work that should have been completed but has not been). An alternative perspective is that the imbalance between workload and capability may occur throughout the project, but that its effect is not apparent until the end (the test phase) of the project. This relates back to a point made in chapter four i.e. that the model of software project schedule behaviour does not distinguish between actual, planned, desired and perceived values of the constructs. Also, this is evidence for the '90% syndrome' i.e. that actual progress is not accurately understood until the work is planned to almost complete.

2. The prevalence of waiting toward the end of the project suggests both a reason for the poor progress toward the end of the project, and an effect of this poor progress. The delays in completing work earlier in the project prevent effort being directed at subsequent work, and capability reduces leading to poor progress. Outstanding work results, and waiting occurs. The waiting subsequently affects capability, during the end of the project. This relates back to a point first recognised in chapter four i.e. the feedback relationships present within the integrated model (and the individual models).
3. The prominence of the design/code and test process areas suggest where the relationships between progress, outstanding work and waiting are most sensitively felt within the two projects. Recall, however, that Project B is more influenced by external process areas than Project C. Also, note that the planning and requirements-gathering phases of the projects has not been investigated.
4. Consistent with the design/code and test process areas being both the most 'sensitive' and the largest process areas of the project, the most frequently reported types of work in the two projects relate to these two process areas. This suggests something of the *content* of the integrated model i.e. that:
   - It is the design and test workload that is more likely to increase.
   - It is the design and test capability that is more likely to reduce.
   - Waiting is more likely in the design and test process areas.
   - Outstanding work is more likely in the design and test process areas.
   - Poor progress is more likely in the design and test process areas.

   Recall, however, that a frequently reported type of work for Project B was decision-making, a type of work not frequently reported for Project C. Also, recall that Project B is often influenced by external process areas, an influence not experienced by Project C (with the exception of the year-2000 requirements).
5. An increase in workload, through the introduction of new features or design changes, not only leads to attempts to increase capability, but also leads to attempts to decrease workload (for example, through reducing the number of testcases, and through prioritising and categorising defects and only fixing a subset of all defects).

9.6 Other studies of actual progress

Actual progress of phases

Watson ([134]) reports on the use of COCOMO ([13, 14] see also, more recently, [12]) as a schedule prognosis and validation tool for a software development project also at IBM Hursley Park. At three points in the progress of the project, the project's

management used COCOMO to estimate the duration of the project and its phases. Unlike Projects B and C, Watson's project completed very much later than planned.

The project in Watson's study is comparable to Projects B and C. The projects come from the same organisation, although Watson's project is slightly larger in terms of code size (original estimate was 81KLOC; actual was 125KLOC) and larger in terms of effort (original estimate was 2592 person-months; actual was 3232 person-months). The Project Leader's opinion of his project is not recorded, but using the joint criteria of project duration and product delivery the project would be judged as unsuccessful, because the project completed very much later than originally planned (planned duration was approximately two years; actual duration was approximately three years.)

The actual progress of the phases in Watson's project are, in some respects, similar to the actual progress of the phases in Projects B and C. Originally the phases of Watson's project were planned as discrete, sequential phases but actually occurred as concurrent phases. The plan phase overlapped with the design/code phase; the design/code phase overlapped with the functional verification and system test phases; the functional verification phase overlapped with the system test phase. All of these phases also actually took much longer than originally planned.

Where Watson's project differs from Projects B and C, in terms of actual progress, is that the test phases (i.e. functional verification and system test) could not start when originally planned. This suggests that the design/code phase was experiencing particularly difficult problems; problems more challenging than those experienced by Projects B and C. For Projects B and C, although the design/code phases were not completed when planned, a sufficient amount of work was completed to allow the test phase to commence. Project C provides a particularly good example: as part of the first internal re-plan, the work within the design/code phase was deliberately re-ordered so that some work (the OS/2 and DOS work) would be completed in time for the planned start of the test phase.

Watson identified a number of factors that may account for the difference between the three COCOMO estimates. These may also account for some of the apparent problems within the design/code phase:

1. As the project milestones slipped, some of the initial requirements became invalid and some function was changed. Projects B and C (both much shorter in duration) both experienced an increase in workload. Although some of the minor requirements for the two projects became out-dated, none of the major requirements became out-dated.

2. The complexity of some parts of the project were underestimated and some parts had to be completely rewritten. In both Projects B and C, there are indications of significant re-designs.
3. The development of the product was dependent on an operating system, which was itself still under development. The project developing the operating system completed late and this affected the progress of Watson's project. Project B was dependent on the delivery of software from another project, but unlike Watson's project this software was delivered when planned.
4. Turnaround time for compilations was assumed to be almost instantaneous when they actually took three to four hours. This is an example of a low-level process affecting the higher-level processes and implies support for the arguments of Bradac *et al.* ([18]).

Watson's study appears to be the only study that has described the actual progress of software projects at the phase-level. Similar patterns of work are available at a lower level of the project. Van Genuchten ([128]), for example, has found a large proportion of activities complete late, and that the late completion is due to the introduction of unplanned activities (i.e. increased workload) and the unavailability of designers (reduced capability). (See chapter two for more detail.)

Rodrigues and Bowers ([103]) use system dynamics models to explore the behaviour of projects. They write:

> "... parallel activities typically have implicit inter relationships which tend to increase the activities' durations, prompting a revision of the plan to incorporate yet more parallelism in an attempt to avoid an overrun." ([103], p. 215)

This relates to the concurrency of phases and the possible effect of this concurrency on the subsequent progress of a project. It might also help to explain the prevalence of waiting, poor progress and outstanding work in projects, where the prevalence of waiting etc. during the end of the project reflects the build-up of parallelism over the duration of the project.

It is particularly unfortunate that there are no appropriate studies of successful projects because these might strengthen the explanation provided above, as well as the wider applicability of these explanations. Phan *et al.* ([96]), for example, report on the development of OS/400, an IBM mid-range operating system. The development was considered an outstanding success, but unfortunately for this study Phan *et al.* did not

present information on phases and process areas. The lack of studies of successful projects relates back to an observation made by Carmel, and first quoted in chapter one:

> "It should be noted that nowhere does the software engineering literature make any causal claims regarding cycle time. Instead, the variables are normative and prescribed for 'successful development'." ([20], p. 112)

Appropriate studies of successful projects would allow causal claims of both unsuccessful and successful development, even if these claims were speculative and required subsequent validation.

Actual workload

There is a large body of research that can be related to the concept of workload. Much of this research uses lines of code (e.g. [129]) or function points (e.g. [4]) as measures of workload, although features (e.g. [121]) and modules ([69]), amongst others, are also used. In addition, much of this research is concerned with developing and/or validating predictive systems, where product size tends to be the main predictor (e.g. [29, 50, 60, 122, 129]). Other studies have sought to provide descriptions and explanations of specific projects. Kornreich and Smith Parker ([63]) report on a case study of the development of a large software system that examined the impact of 127 requirements changes on project duration. They found that the additional requirements account for several additional months of work. Mouakket *et al.* ([80]) report on a case study of the development of a small software system, again examining the impact of requirements changes. They found that many of the original requirements were not implemented in the final product, being replaced by requirements that evolved during the duration of the project.

All of these studies use some concept of workload, and a logic that changes in workload impact the duration of a project. (There are differences of opinion, across these studies, as to the exact relationships between workload and duration.)

Actual capability

The premise of software process modelling and improvement is that improving the way the software is developed (the process) will improve the performance of the project (e.g. cost, effort, duration) and the quality of its output (i.e. the software system produced). Based on this premise, software process modelling and improvement is directly concerned with capability and improving capability. The Capability Maturity Model (e.g. [88-90])

and the People Capability Maturity Model (e.g. [30]) are well-known examples in this area.

Chapter one excluded much of this research, relying on Rodden *et al.* ([102]) to argue that there was a tendency to develop abstract models of processes (models that may be too abstract), with a lack of real attention directed at actual processes. The evidence and analysis presented in this thesis is an example of the kind of work that is typically *not* conducted by software process research. The implication is that much of the body of software process research is difficult to relate to the research reported on here. The terminology, notations and models being developed by software process research might, however, be usefully applied to analysing actual process and the evidence presented in this thesis. This stands as an opportunity for further research.

Watson's study ([134]), discussed earlier, hints at capability issues but does not consider them explicitly (recall that Watson identified dependencies with another project and poor compiler turnaround time as factors which, in this context, are capability issues). Watson's study serves as an example that studies do detect process inefficiencies or process problems, but do not model them explicitly as capability issues.

9.7 Other studies of the characteristics of process areas

Much of the available research on waiting has already been reviewed in chapter two and related to the current investigation in section 9.3. With regards to the analysis of reports of poor progress and reports of outstanding work, there appears to be little, if any, relevant research previously conducted within the software engineering research community. Bradac *et al.* ([18]) and Perry *et al.* ([92]) express an interest in the progress of work, but do not pursue that interest. Also, research on the '90% syndrome' might be related to evidence on poor progress.

9.8 Other studies of the tactics of management

Although not explicitly drawing on empirical evidence, Rodden *et al.* ([102]) argue that:

> "All organisational life involves 'cutting corners', informal 'bending of rules' and so forth. In most instances, organisational managements are aware that such work goes on, if not in detail, and allow it precisely because *it is a means by which the work can be done*." ([102], p. 61; emphasis added)

Tactics of management are means by which the work gets done. They are essentially pragmatic and informal because they respond to unpredictable contingencies, interruptions and problems which arise as work is undertaken in practice ([118]). It is clear from this quotation that Rodden *et al.* consider this kind of behaviour to occur in all organisations and, by implication, in many if not all projects. As already stated, however, they do not provide explicit empirical evidence to support their claim. In a subsequent paper, Sommerville and Rodden ([118]) report on two case studies, making observations that are more directly comparable to the tactics of Projects B and C. They write:

> "... the practical reality is that the actual work done and the way in which it is done is continually re-negotiated at a very detailed level by the participants themselves." ([118]; p. 6)

and in so doing they echo the words of Project C's Project Leader:

> "We are constantly juggling work assignments to even the workload." [Interview C.007.CP]

Similar observations are made by Waterson *et al.* ([133]) who conduct a case study of the impact of cognitive and organisational factors upon the work of a commercial software development project. Waterson *et al.* observe that workload fluctuated and that teams would be temporarily restructured (with staff being drafted in from other teams in the project if the workload became too demanding) to ensure that project milestones and deadlines were met. Furthermore, Waterson *et al.* conclude that one of the major successes of the project is its *ability to reallocate and re-negotiate tasks and responsibilities*. This supports an argument made earlier in this thesis i.e. that the socio-technical context of a project might delimit the scope within which tactics of management can be effectively employed. In Waterson *et al.*'s study, tactics of management could be successfully employed because the project was not (too) restricted.

As explained in chapter two, Perry *et al.* ([92]) recognise that developers may be reassigned to higher priority projects, and for them the reassignment of developers to other work reflects the fact that the organisation of large-scale software development projects is extremely dynamic. Also reviewed in chapter two, van Genuchten ([128]) found that delays and overruns to activities increased toward the end of the project, but he uses this observation to *discourage* the use of tactics of management later in the project.

McKeen ([76]) investigates the development profiles of 32 software projects, all developing business applications, across five organisations. He recognises that completion deadlines are subject to some "manipulation" (McKeen's phrase); a behaviour clearly observed in Projects B and C. McKeen also observed that similar manipulations are not possible for cost and effort, because the time reporting systems in these organisations make it difficult to arbitrarily adjust the actual effort and therefore the actual cost. Such constraints do not appear to apply to Projects B and C. For example, because of the costing approach for Project B (where Project B is costed together with three other projects), some resource is drawn away from the subsequent project to support Project B. Similarly, with Project C, although much of the resource (and hence cost) is allocated to support work, that resource is assigned to new development work. The manipulation of phases, resource allocation and cost are all examples of tactics of management.

To summarise, the tactics of management are pragmatic, informal and often 'hidden' methods for getting work done on projects, and are concerned with 're-shaping' work and process. They may be constrained in their application by the project's surrounding socio-technical context.

9.9 Other studies of the socio-technical contexts of projects

The significance of organisational influences on the progress of a project is widely recognised by both researchers and practitioners. Sommerville and Monk ([116]) consider that the response of a software development manager and software engineers to some event within their project is not only determined by the problem but by wider organisational factors. Block ([11]) argues that the external component of a project is the major contributor to that project failing. Quintas ([98]) argues that the adoption of software engineering is mediated and resisted by social, organizational, cultural, and institutional factors. Also, some research into software estimation argues that prediction systems must be calibrated to the environments in which they will be used (e.g. [29, 56]).

Thamhain and Wilemon ([123]) have looked at the problems that make projects difficult to control and from those they provide some recommendations for controlling projects. One particular recommendation is to assure continuous senior management involvement, endorsement and support of the project. The strategic value of Projects B and C relate to this issue. Because of the higher value of Product B, it appears that Project B has the support of senior management. Project C did not appear to be supported to the same degree. (Since the completion of Project C, Product C has been moved into a new business division of the organisation and is now receiving more support from their new senior management.)

Research is also recognising the need to distinguish different types of development for different types of product. For example, Jackson (e.g. [52-54]) and others (e.g. [109]) argue that just as the more traditional disciplines of engineering are distinct professions (e.g. civil engineering, electronic engineering, mechanical engineering), so software engineering will evolve into distinct areas of specialist knowledge, based on distinct and well-defined problem domains. These different problem domains, by definition, present their own intellectual and technological issues.

Similarly, some prediction systems identify types of product and types of project as 'drivers' in the estimation models (e.g. [13] and more recently [12]). Also, some surveys distinguish different types of product and project (e.g. [35]).

9.10 Applying the insights to a wider 'population' of projects

A recurring concern throughout this thesis has been the degree to which one may apply the findings of case study research to a broader set of projects. Paraphrasing Wolcott ([137], p. 173)[6] as a guideline for generalising:

> Every software project is in certain aspects:
> a. like all other software projects
> b. like some other software projects
> c. like no other software project

In generalising the findings from this investigation, a clear distinction should be made between the three models and the numerous insights. The models provide a framework within which the insights have been drawn and organised. Because of the intended flexibility of these models, one would expect them to apply to a wide range of (if not all) software projects. The models propose a small number of generic relationships which are intuitively sensible, and for which there is supporting empirical evidence (e.g. the brief review of supporting research in chapter four, the behaviours of Project B and C, and the review of previous research in sections 9.6 through 9.9). While the relationships are causal, and so form the basis for prediction, the models (in their current form) are not

---

[6] Wolcott ([137], p.173) actually writes:
> Every man is in certain aspects:
> a. like all other men,
> b. like some other men,
> c. like no other man.

This aphorism was first used by Kluckhohn and Murray ([62]).

intended as predictive systems. The relationships appear to apply to all software projects, although they might require (substantial) calibration to particular software projects.

Project B, Project C and Watson's project appear to lie along a continuum of the degree to which a project's actual duration matches the originally planned duration. The actual project duration for Project B is the same as the planned duration. The actual project duration for Project C is slightly longer (a few weeks) than the planned duration. The actual duration for Watson's project is very much longer than the planned duration. Despite these differences, all three projects exhibit similar schedule behaviour, with various phases completing later than planned and progressing concurrently with one another. This suggests that it is common for the schedules of software projects to actually behave differently from that planned, and that project's internally re-plan their schedules in response to changes internal and external to the project.

Combining the insights from Projects B and C with the various empirical studies reviewed suggests that project managers often employ various tactics of management in order to get the work done. Also, the presence and frequency of waiting is common across projects.

A number of differences between Projects B and C immediately suggest parameters for distinguishing populations *viz.* the strategic value of a product, the type of product, the type of project, the nature and structure of project management, and the degree of expected and unexpected external influence.

Overall, the three models, the behaviour of project schedules, the tactics of management and some characteristics of waiting appear to have wide applicability. Elements of the socio-technical contexts provide parameters for distinguishing different types of software project. Certain other characteristics, such as poor progress and outstanding work, lack a sufficient body of accumulated evidence with which to make a judgement on their applicability.

Chapter 10 A summary of the thesis

10.1 Introduction

This thesis opened with the argument that there is an industrial need for a better understanding of the actual behaviour of software projects and, more particularly, of their schedule behaviour. Software project schedule behaviour was defined as the dynamic structure of time and work on the project. Clearly, the general behaviour of the project will affect the project's schedule behaviour.

The thesis then drew upon Eisenhardt to argue that explanations of actual behaviour require an intimate connection with empirical reality. An intimate connection with empirical reality necessarily requires a close and solid connection with the actual processes of particular, real-world software development projects.

The thesis then reviewed studies of actual time usage on software development, as these provide the most direct connection with intervals of time and instantaneous events. This review noted that previous research has concentrated on the lower-level processes of software projects, and that there are few studies that have investigated higher-level processes, related the lower-level processes to the higher-level processes, or related these combined processes to schedule behaviour. These gaps in extant research formed the basis for the subsequent empirical component of this investigation.

Two case studies of real-world software development projects at IBM Hursley Park were conducted (Projects B and C). A large volume of evidence was collected and analysed. Because the general behaviour of the project will affect its more specific schedule behaviour, the case studies took a broad perspective in investigating the projects, considering aspects of the projects that might not immediately relate to the project's schedule behaviour.

The remainder of this chapter summarises the various empirical analyses conducted as part of this investigation; the main conclusions that emerge from these analyses; the recommendations, for research and industry, that follow from the conclusions; the threats to, and limitations of, the investigation; and the opportunities for further research. The chapter concludes with a review of the aims and objectives of this investigation (as outlined in chapters one and two respectively).

10.2 A summary of the components of the empirical analyses

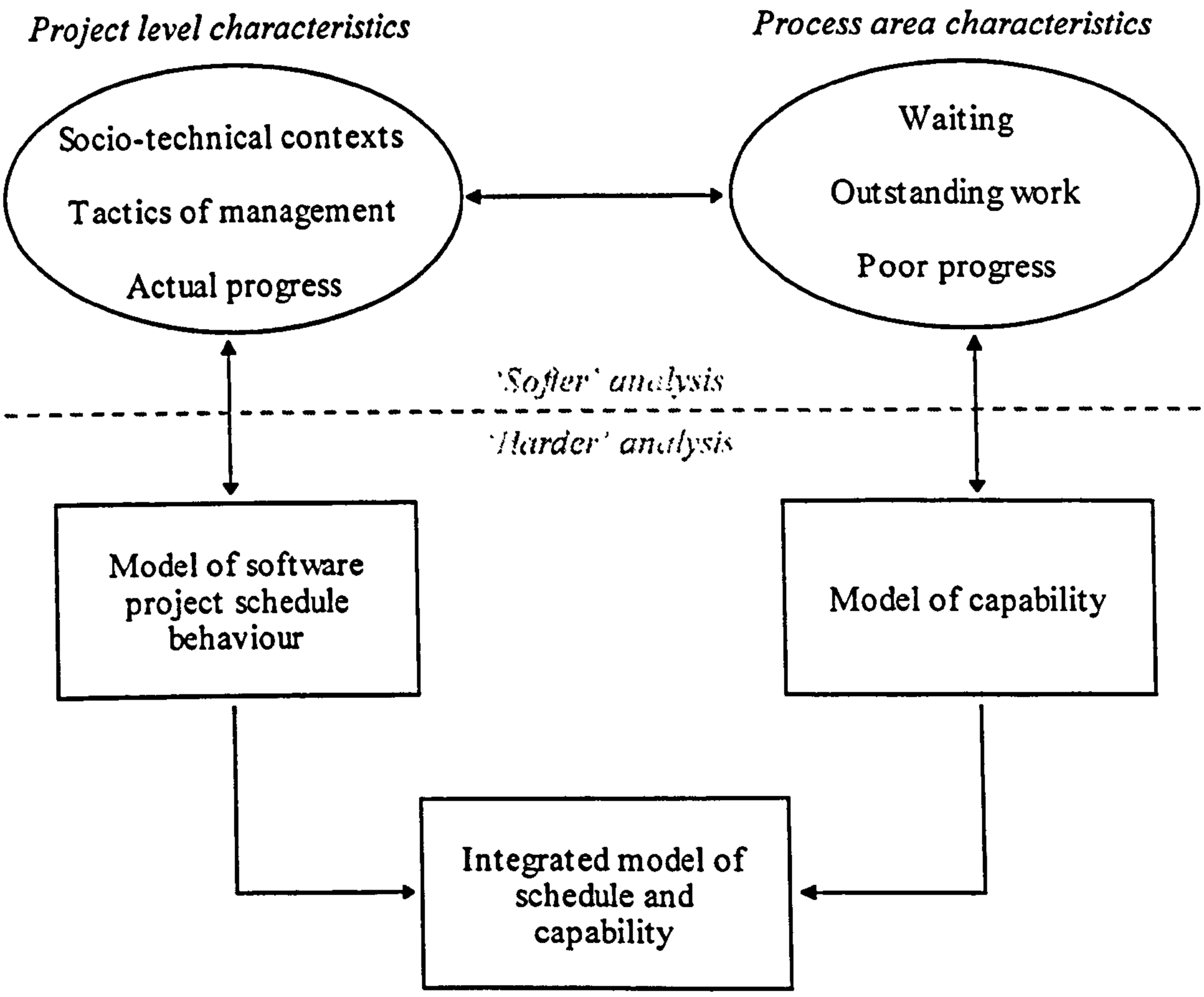


Figure 10.2.1 The components of the empirical analyses

Figure 10.2.1 illustrates the various empirical analyses that were conducted as part of this investigation. The figure distinguishes two bodies of empirical insights and three conceptual models, and indicates how these five components relate to each other. The two bodies of empirical insights are the analysis of characteristics of the project and the analysis of the characteristics of process areas within the project. The three conceptual models are the model of software project schedule behaviour, the model of capability, and the integrated model of schedule behaviour and capability. The figure also suggests two 'modes' of analyses: a 'softer' analysis, where the evidence has been related to each other, but without an explicit model for the comparisons; and a 'harder' analysis, where the evidence has been compared, using the three models as a vehicle for the comparison. The empirical insights have been drawn from both Projects B and C. The three models have each been applied to both Projects B and C. Previous research has been related to the empirical insights and the models.

10.3 Conclusions and implications

Before discussing the conclusions and implications to be drawn from this investigation, it should first be emphasised that while the concept of a project as a distinct entity seems reasonable and is used in both research and industry, the actual boundaries of a project in an organisation's 'space-time' are ambiguous and very difficult to properly define. Projects B and C both provide examples of ambiguity in defining a project.

Conclusions

The first conclusion from this investigation is that projects which complete according to their originally planned duration exhibit internal behaviour, and not just internal *schedule* behaviour, similar to projects that complete later than their originally planned duration. More specifically, for both types of projects:

- The internal schedules are similar:
  - Phases complete later than planned.
  - Phases occur concurrently when they were planned to occur sequentially.
  - The major milestones occur toward the end of the project.
- There are explicit and implicit increases in workload.
- There are implicit decreases in workload.
- There are implicit increases in capability.
- There are similar tactics of management.
- There are similar characteristics of waiting, poor progress and outstanding work.

There are notable *differences* in the socio-technical contexts of the two types of projects, particularly:

- The (relative) strategic value of the product.
- The type of product.
- The type of project.
- The methods of managing the project, in terms of:
  - The structure and purpose of status meetings.
  - Some strategies for managing the project.

It may be that although both types of project experience explicit and implicit increases in workload, projects that complete according to their originally planned duration are better able to implicitly decrease workload and/or to implicitly increase capability. (Waterson *et al.* [133] drew a similar observation from their case study.) The ability to respond to

changes in workload is conjectured as being affected by aspects of the socio-technical context of the project (as delimited above). As an example, Project B was costed as part of a set of four projects. As the resource is funded across four projects, it might be easier to 'borrow' some of that resource from an associated project, because there is no change in the overall cost of either project. This tactic might further be justified with the argument that Project B is delivering some work originally intended to be delivered with the subsequent project (see chapter five for more detail) and so is 'justified' in borrowing resource from that project.

A second conclusion is a strengthened confidence in Bradac *et al.*'s conjecture that waiting is more prevalent during the end of the project than during the middle of the project. The characteristics of waiting, poor progress and outstanding work all support this conjecture. In addition to strengthening the confidence one may place in Bradac *et al.*'s research, there is also a strengthened confidence in the studies that build upon Bradac *et al.*'s research (i.e. [5, 34]) because this investigation has tested assumptions made by those subsequent studies.

A third conclusion is that the three models appear to be useful for describing and explaining the behaviour of software projects in general and software project schedules in particular. With further research, these models might evolve into theories and/or predictive systems.

Fourth, this investigation also provides complementary evidence regarding actual time usage within projects. In contrast to a number of previous studies that have focused on the lower-level use of time (e.g. how individuals use their time), this investigation has studied the use of time at higher-levels of the process (i.e. how process areas and the project itself uses time).

Finally, empirical support from a number of other studies complement the insights gained in this investigation and increase the confidence with which one can apply some of these insights to other projects. Specifically:

- The three models, the project's schedule behaviour, the tactics of management and some characteristics of waiting appear to have wider applicability.
- Elements of the socio-technical contexts provide potential parameters for distinguishing different types of software project.

Certain other insights, however, such as those referring to the characteristics of poor progress and outstanding work, lack a sufficient body of accumulated evidence with which to make a judgement on their wider applicability.

Implications

Two main implications can be drawn from the above conclusions. First, despite the approach taken in this investigation (i.e. the collection of a large volume of evidence and the analyses of a wide variety of factors using a very broad perspective), this investigation has been unable to pinpoint definite causes to explain why a project will or will not complete according to its original plan. The only 'hint' of an explanation are the differences between the socio-technical contexts of the two projects and, related to this, the fact that tactics of management may be constrained by a project's socio-technical context.

The second implication is that this research re-directs attention toward the investigation of those processes that surround a project and not just those processes that occur within the project. (Section 9.9 identifies some studies that have looked at processes broader than just the project.) This relates back to the issue of the ambiguity of defining a project. It may be that those things that make a project difficult to distinguish from its surrounding organisation are precisely those things that explain the progress of that project. This seems to be apparent in the fact that the tactics of management either exploit such ambiguity or at least work within it e.g. Project B borrowing resource budgeted to Project B+1.

10.4 Recommendations

Table 10.4.1 A summary of recommendations for industry

#	Recommendation
1	Model and improve the processes within and between the design/code and test process areas, as these appear to be the most sensitive and largest process areas within a software project.
2	Model and improve the processes relating to design/code and test work as these appear to be the most significant types of work within a software project. (*cf.* recommendation #6)
3	Distinguish different types of process models (and perhaps plans) and process improvement programmes based on elements of the socio-technical contexts, particularly: • The type of product. • The type of project. • The strategic value of the product. • The structure of the status meetings.
4	Focus improvement on the planning process, particularly on the role of the plan during the execution of the project.
5	Model and improve the internal re-planning process.
6	Focus process improvement on inter-project processes and not just intra-project processes. In particular, focus on inter-project processes relating to the design/code and test process areas and the design/code and test types of work. (*cf.* recommendation #2)
7	Model the dependencies between activities, within and between projects, as these are sources of waiting, and may threaten capability.
8	Model the dependencies of activities on resource as these are sources of waiting, and may threaten capability.
9	Improve the methods of reporting progress, particularly during the design/code phase, through: • metrics. • structured status meetings.
10	Introduce effective milestones during the design/code phase. This is dependent on the introduction of effective reporting of progress (*cf.* recommendation #9).
11	Identify and communicate tactics of management and the circumstances within which a tactic is applicable.

Table 10.4.1 presents some recommendations for practitioners, based on the insights gained during this investigation. Most of the insights refer to modelling and improving the software production and management processes of software projects.

Recommendation #2 is distinct from recommendation #1 because processes relating to design/code and test work may not just occur within the design/code and test process areas. As is clear with Project B, dependencies exist with other projects due to design/code and test work (see recommendation #6).

Table 10.4.2 A summary of recommendations for research

#	Recommendation
1	Conduct research into actual process, considering: • Processes at a variety of levels of the project. • Processes within and between a variety of process areas, particularly the design/code and test process areas. • The interaction between processes at various levels. • The short-term and long-term effects of processes. • Both the production process and the management process.
2	Investigate the development and application of different types of process models and process improvement programmes for different types of project, product and socio-technical context.
3	Investigate mechanisms for reporting progress, whether these be improved metrics or more structured methods of reporting.
4	Investigate the actual flow of work through a project, particularly relating to the design/code and test process areas.

Tables 10.4.2 presents some recommendations for the *focus* of research, based on the insights gained from this investigation. These recommendations are distinct from recommendations on how to *conduct* research (*cf.* the heuristics presented in chapter three). Also, the recommendations presented in Table 10.4.2 are more general recommendations, in contrast to more specific recommendations presented in Table 10.6.1

10.5 A critical review of the investigation

Table 10.5.1 A summary of threats to the investigation

#	Threat	Chapter
1	The review of research has concentrated on research within the software engineering community. This may exclude valuable studies from other areas of research.	Two
2	It is difficult to generalise from case study research: • Practical constraints limit the depth of inquiry (in terms of the degree to which each case can be investigated) and the breadth of inquiry (in terms of the number of cases that can be considered) This may distort the applicability of the insights and models to other projects.	Three
3	The status meeting minutes may not provide a reasonable representation of the progress of the project. Also: • Meeting minutes were simplifications. • Meeting minutes did not occur for every week of the project (*cf.* #13). This may distort the test of Bradac *et al.*'s conjecture, the insights drawn about the characteristics of waiting, and the value of the model of capability.	Three
4	The status meeting minutes for Project B were from the project status meetings, but for Project C the minutes were from the design/code/test status meetings. This may distort the comparisons between the two projects.	Three

Table 10.5.1 A summary of threats to the investigation (*continued*)

#	Threat	Chapter
5	The definitions of the beginning, middle and end of a project may not match the definitions used by Bradac *et al.* This may prevent a proper replication of Bradac *et al.*'s work.	Three
6	There was no investigation of the requirements and planning phase of the project. This prevents a complete replication of Bradac *et al.* Also the success of the project may be influenced by the progress of the planning phase.	Three
7	With regards to the analysis of waiting, the evidence was only analysed for references to waiting and not for references to working or not working. This might interfere with a proper replication of Bradac *et al.*'s work.	Three
8	The phrases used in the searches for references to waiting, progress of work and outstanding work were not exhaustive, in that they did not contain all of the different kinds of terms that could refer to waiting, progress of work and outstanding work. This may interfere with the evidence representing the phenomena of interest i.e. waiting, progress and outstanding work.	Three
9	There was no measurement of the size of the effect of poor progress, outstanding work and waiting in the model of capability.	Three
10	The model of software project schedule behaviour has not been formally validated. This might distort the value of the model for explaining behaviour, and potentially predicting behaviour.	Four
11	The model of capability has not been formally validated. Again, this might distort the value of the model for explaining behaviour, and potentially predicting behaviour.	Four
12	The complexity of the behaviour of the projects threatens the valid description and explanation of their behaviour.	Five
13	The lack of weekly status meetings threatens the consistency of the waiting, progress of work and outstanding work evidence (*cf.* #3).	Six
14	Reports of waiting, progress and outstanding work may not be representative of actual waiting, progress and outstanding work. This may interfere with the evidence representing the phenomena of interest, and subsequently the validity of the insights drawn from the evidence.	Six
15	There is a methodological difference between Bradac *et al.*'s study and this study, in terms of what behaviour could be observed. Bradac *et al.*'s study consisted of direct observations of behaviour, whereas this study observed behaviour indirectly through the meeting minutes. This methodological difference may prevent a proper comparison of the two studies.	Six
16	The influence of the tactics of management, possible time delays between cause and effect, and the influence of feedback, threaten the value of the models for explaining behaviour	Nine

Table 10.6.1 presents a summary of the threats to, and limitations of, the investigation. The table also indicates the primary chapter that the threat relates to.

10.6 Opportunities for further research

Table 10.6.1 A summary of the opportunities for further research

#	Opportunity
0	**Overall**
0.1	Replicate this investigation.
0.2	Conduct complementary investigations: • Surveys would address concerns of wider applicability. • Experiments would formally validate (in terms of hypothesis-testing) aspects of the models and the findings.
0.3	Investigate the relationship between the design and test process areas.
4	**Arising from chapter four**
4.1	Develop the model of software project schedule behaviour: • Distinguish degrees of change in workload, capability and remaining-duration. • Identify and model the processes that impact workload and capability. • Distinguish between actual, desired, planned and perceived values of remaining-duration, workload and capability. • Apply the model to various aspects of a project e.g. at the project level, at the process-area level, to a particular feature, for a particular team. • Develop the model as a feedback system.
4.2	Validate the model of software project schedule behaviour: • Through additional empirical studies. • Through comparison with existing empirical evidence.
4.3	Develop the model of capability.
4.4	Validate the model of capability: • Through additional empirical studies. • Through comparison with existing empirical evidence.
5	**Arising from chapter five**
5.1	Investigate the nature of the socio-technical contexts of projects, and the effects these contexts have on workload, capability and remaining-duration.
5.2	Investigate the internal behaviour of successful and unsuccessful projects.
5.3	Investigate the tactics of management.
6	**Arising from chapter six**
6.1	Investigate the effect of concurrent phases on the prevalence/frequency of waiting.
6.2	Investigate the knock-on ('second-order') effects of source and dependent process areas.
6.3	Investigate the feedback relationships between source and process areas.
7	**Arising from chapter seven**
7.1	Investigate the effect of concurrent phases on the progress of work
9	**Arising from chapter nine**
9.1	Investigate the frequency of waiting, poor progress and outstanding work per process area per week.

Table 10.6.1 summarises the opportunities for further research that have been identified in this thesis. The summary is organised according to the chapter in which the opportunity was identified. As is clear from the table, there are a considerable number of

directions in which subsequent research on software project schedule behaviour may progress.

A particularly important direction for further research would be to further investigate both successful and unsuccessful projects. Furthermore, such investigations should examine the actual behaviour of the projects, and examine that behaviour in detail. This would naturally suggest the conduct of further case studies, but the careful design and administration of survey studies might also provide valuable evidence. Survey studies would be particularly valuable if they could provide evidence across a relatively large number of cases; a necessary pre-requisite for generalising these conclusions.

10.7 A review of the aims and objectives of this investigation

Chapter one presented four aims to this investigation, and chapter two presented three specific objectives. These aims and objectives are re-presented here, together with brief comments on the degree to which they were satisfied in this investigation. The aims are (specific objectives are included as part of the third and fourth aims):

1. *To consider the degree to which existing empirical studies within the software engineering research community identify, describe or explain relationships between the actual processes of software development and the schedule behaviour of software projects.*

   Chapter one and the opening sections of chapter two argued that there is a lack of research, within the software engineering community, that seeks to *generate* explanations of software project behaviour in general, and software project schedule behaviour in particular. Five bodies of research were briefly considered in making this argument:

   - Surveys of practitioners' opinions of the software process.
   - The development and validation of system dynamic models of software development projects.
   - The development and validation of prediction systems of characteristics of software projects e.g. effort, cost, quality and duration.
   - The development and validation of software process models.
   - Investigations of actual process.

   Studies of actual time usage in software development projects were identified as the most likely sources of research to identify, describe and explain how actual processes

relate to software project schedule behaviour. This is because these studies explore both 'visible' and 'invisible' work, and because they provide the most direct connection with intervals of time and instantaneous events in a software project.

2. *To identify gaps within the existing research that prevent, or limit, the development of a theory.*

The review in chapter two concluded that there are a lack of studies of higher-level processes, of the interaction between lower-level and higher-level processes, and of the effect of these two sets of processes on software project schedule behaviour. The chapter also concluded that empirical and theoretical knowledge within this area has not developed to the extent that one can forward a valid, testable and relevant theory of software project schedule behaviour.

3. *To identify the opportunities for a contribution in this area of research, and to select one or more of these opportunities as specific objectives for the empirical component of this research.*

Chapter two identified three specific objectives for the empirical component of this investigation. They are:

- *To replicate parts of Bradac* et al.*'s study.*
  Bradac *et al.*'s conjecture that waiting is more prevalent during the end of the project than during the middle of the project was tested with six sets of evidence, three sets from each of the two projects. Furthermore, Bradac *et al.*'s observations of types of waiting were also explored, but this investigation found a different set of types of waiting to those identified by Bradac *et al.*

- *To investigate actual time usage at higher-levels of the project.*
  The two case studies explored the socio-technical contexts, the actual progress, and the tactics of management of projects, and the waiting, progress of work and outstanding work characteristics of process areas.

- *To investigate the relationships between the lower-level and higher-level processes, and their relationships to schedule behaviour.*
  The two case studies sought to relate the characteristics of the project to the characteristics of process areas (see chapter nine). Also, the model of software project schedule behaviour, a model of higher-level processes, was integrated with

the model of capability, a model of lower-level processes. The integrated model was also related to the studies of actual time usage reviewed in chapter two.

4. *To conduct empirical inquiry, so as to contribute to the body of research on software engineering in general and software project schedule behaviour in particular.*

Broadly, four contributions were made:

1. The development of three models i.e. the model of software project schedule behaviour, the model of capability and the model of schedule behaviour and capability.
2. The drawing of a number of insights concerning software project behaviour in general and software project schedule behaviour in particular. These insights refer to the socio-technical contexts, the actual progress, and the tactics of management of projects, and the waiting, progress of work and outstanding work characteristics of process areas.
3. The drawing of a number of more general conclusions and implications concerning software project behaviour. These conclusions principally refer to the difficulty in distinguishing between the behaviours of successful and unsuccessful projects.
4. The generalisation of these insights to other projects, through:
   - A review of previous research in light of the insights drawn from the case studies.
   - An independent test of a conjecture of Bradac *et al.*'s.

Glossary

Term	Definition
Beginning of the project	The period between the start of the planning phase and the start of the design/code phase of a project. See chapter three for more information.
Capability	Broadly, the ability to complete the work in the project. A more technical definition of capability is the ability to complete *n* units of work per unit time, at time *t* of the project. See chapter four for more information.
Defect screen team	A team that categorises, prioritises and allocates defects to defect-fixers.
Design changes	A set of market requirements of a piece of software which typically involve changes and additions to multiple software subsystems. (See also features.)
End of the project	The period between the start of the test phase of the project and the end of the project.
Feature	A set of market requirements of a piece of software which typically involve changes and additions to multiple software subsystems. Features and design changes are conceptually similar. Features tend to be design work that is recognised and planned for during the initial planning phase. Design changes tend to be work that is introduced as the project progresses.
Global process	See higher-level process
Higher-level process	A process occurring: • within a process area • between process areas • between projects, or • between the project and the organisation.
KLOC	An acronym for thousands of lines of (software) code, a measure of the size of a software product.
Laboratory	Except where indicated otherwise, this term refers to IBM Hursley Park.
Lower-level process	A process occurring within an individual (e.g. cognitive processes), between individuals (e.g. communications, such as emails), or between teams.

Term	Definition
Middle of the project	The period between the start of the design/code phase and the start of the test phase.
Organisation	Except where indicated otherwise, this refers to IBM Corporation.
Outstanding work	Work that should have been completed but which has not been. See chapter four for more detail.
Plan	A project plan defines: the project objectives, the necessary work to achieve those objectives, when and by whom this work will be performed, the methods to be employed, how long the project will take, and how much it will cost. (taken from [75], chapter 27 page 27)
Process area	A 'production unit' of the project, such as the Design/Code process area (which produces the designs and software code) or the Test process area (which tests the designs and software code).
Product area	The management and production areas concerned with the development and subsequent support of a product. The product area is 'wider' than the project. Project B, for example, is one project within Product B's product area.
Progress of work	An indicator of the 'imbalance' between workload and capability, such that the workload may not be completed with the current capability in the duration planned. See chapter four for more information.
Project Assistant	The Project Assistant assists the Project Leader and the project management team in managing the project. The Project Assistant is primarily an administrative role.
Project Leader	The most senior individual (the individual with the highest authority) within the project. The Project Leader heads the project management team of the project.
Project management team	A multi-functional management team, consisting of representatives from all the (important) process areas of a project.

Term	Definition
Remaining duration	Technically, the amount of time remaining on the project, at time *t* of the project. (See the model of software project schedule behaviour in chapter four.)
Software project schedule behaviour	The dynamic structure of time and work on a project. See chapter one for more detail.
Waiting	Waiting occurs where one process area is waiting on another process area for the delivery of some resource (e.g. code, or the availability of personnel). See chapter four for more information.
Work breakdown structure (WBS)	A product-oriented task hierarchy of all the work to be performed to accomplish the project contractual objectives. The products may be elements of software, hardware, documents, tests, reports, support services, or other quantified elements of the objectives. (Taken from [75], chapter 27 page 20)
Workload	Broadly, workload is the amount of work to be done on the project. Technically, workload is the number of units of work remaining to be completed, at time *t* of the project. See chapter four for more information.

Appendix A0 Deciding meaningful associations between entities

A0.1 Introduction

A number of tables in the main body of the thesis have been used to draw conclusions about the meaningfulness of associations between entities. For example, in chapter six, Table 6.5.2 was used to draw conclusions about the associations between 'source' process areas and 'dependent' process areas.

The first method considered to determine meaningful associations between entities was to use a Poisson distribution to calculate the probabilities of random allocations of items to cells in a table. Those allocations which were unlikely to occur randomly where considered to be meaningful. A fundamental problem with this procedure, however, is that to use a Poisson distribution one must be able to reasonably assume that the allocation of each item occurs *independently* of the allocation of each and every other item. For each table in this thesis, however, there is a finite number of items that can be distributed between the cells of that table. Consequently, once an item is assigned to a cell, it affects the probabilities of randomly assigning the remaining items to cells. (The simplest example is a situation where one has two cells and one item to assign to those cells. If the item is assigned to the first cell, it cannot be assigned to the second.) Thus, one cannot reasonably make the assumption of independence, and so one cannot employ a Poisson distribution.

Section A0.2 documents this first procedure. Equation 4 cannot be retained because of the inability to assume independence. Because the logic of the first procedure is still valuable, it was used as a basis to develop a second procedure. The second procedure comprised a number of computer simulations, conducted to calculate the probabilities of randomly assigning items to cells of a table, where each assignment is not independent of previous assignments. This procedure is described in section A0.3

A0.2 Using a Poisson distribution to decide meaningful associations

In a Poisson distribution, λ (lambda) is the mean number of occurrences per 'grouping'. Typically, λ is the mean number of occurrences per interval of time, but a Poisson distribution can also be used for other kinds of processes ([64]). In this thesis, λ is the mean number of associations between two entities e.g. the mean number of associations between 'source' process areas and 'dependent' process areas in Table 6.5.2.

The following procedure can be used to calculate the probabilities of assigning *n* items, or higher, to a cell in table *where the probabilities of assigning each item are independent of other assignments*:

1. Calculate λ:

$$\lambda = \frac{T}{C} \qquad \text{[Equation 1]}$$

where *T* is the total count of all the items in the table and *C* is the total number of cells for that table, excluding items and cells related to 'Unknown' categories (see section A0.5 for further information).

2. Calculate the probability, P(*n*), of *n* items occurring in a cell (formulae taken from [72]):

$$P(n) = \frac{\lambda^n e^{-\lambda}}{n!} \qquad \text{[Equation 2]}$$

3. Calculate the probability, P(<*n*), of less than *n* items occurring in a cell:

$$P(< n) = \sum_0^{n-1} P(n) \qquad \text{[Equation 3]}$$

4. Calculate the probability, P(<*N*), of less than *n* items occurring in all cells in the table:

$$P(< N) = (\sum_0^{n-1} P(n))^C \qquad \text{[Equation 4]}$$

5. Calculate the probability, P(≥N), of one or more cells having at least *n* items:

$$P(\geq N) = 1 - (\sum_0^{n-1} P(n))^C \qquad \text{[Equation 5]}$$

6. If *P*(≥*N*) is less than or equal to 0·05 then the value is considered significant and any associations, with a value of *x* or greater, between entities are considered meaningful. If *P*(≥*N*) is less than or equal to 0·01 then the value is considered very significant and any associations, with a value of *x* or greater, between entities are considered meaningful. If *P*(≥*N*) is less than or equal to 0·001 then the value is considered

extremely significant and any associations, with a value of x or greater, between entities are considered meaningful.

A0.3 Using a series of computer simulations to decide meaningful associations

As already noted, a number of computer simulations were conducted to estimate the probabilities for the random assignment of items amongst a number of cells in a table. Because meaningful associations were being estimated for a number of tables, and each table had a different number of items to allocate and a different number of cells, a number of computer simulations were conducted. Each simulation consisted of running 20 sets of 50,000 runs, where each run made one estimate of the probabilities of each allocation occurring. For each set of 50,000, the probabilities were averaged. The final averages for each of the 20 sets were then used to calculate a mean probability, and a standard deviation from the mean. The standard deviation was used to set the confidence level, ± 2 standard deviations, for the mean estimates. Figure A0.3.1 presents a copy of the source code for the computer program used to simulate the distributions. (Sincere thanks to Colin Kirsopp for writing and testing this program.) Parameters input to the program are presented and discussed in section A0.4.

Table A0.3.1 Input parameters to the program

Parameter	Comment
Label	A text description of the simulation. For referencing purposes.
CellCount	The number of cells to which items can be randomly allocated.
ItemCount	The number of items to randomly allocate amongst the cells.
RunCount	The number of runs of the allocation. Each run will randomly allocate all the items to the cells.
SetCount	The number of sets of runs. Using a number of sets allows one to estimate a confidence interval for the calculation of probabilities.
MaxValue	The highest value to calculate and record the probability for. Probabilities are calculated and recorded for values between 0 and MaxValue.

The source code for the C++ program used to simulate the random allocation of items to cells in a table is listed below. The program receives six inputs which are identified and described in Table A0.3.1. Each input should be on a separate line of the input file. Section A0.4 presents a full list of the inputs for all of the simulations.

Figure A0.3.1 Source code for the computer program used to simulate the distributions

```
#pragma hdrstop
#include <condefs.h>
#include <stdlib.h>
```

```
#include <stdio.h>
#include <iostream>
#include <fstream>

//---------------------------------------------------------------------
#pragma argsused

int main(int argc, char **argv)
{
    int cellCount, itemCount, setCount, maxValue;
    long runCount;
    long* cells;
    long* lessThanX;
char* inputBuffer = new char[202];

if(argc != 2)
{
    cout << "Usage:" << endl;
    cout << "PoissonTest datafile" << endl;
    exit(1);
}
ifstream fin(argv[1]);
while(!fin.eof())
{
    randomize();
    fin.get(inputBuffer, 200, '\n'); fin >> ws;
    char* label = new char[strlen(inputBuffer)+1];
    strncpy(label, inputBuffer, strlen(inputBuffer));
    fin.get(inputBuffer, 100, '\n'); fin >> ws ;
    cellCount = atoi(inputBuffer);
    fin.get(inputBuffer, 100, '\n'); fin >> ws ;
    itemCount = atoi(inputBuffer);
    fin.get(inputBuffer, 100, '\n'); fin >> ws ;
    runCount = atol(inputBuffer);
    fin.get(inputBuffer, 100, '\n'); fin >> ws ;
    setCount = atoi(inputBuffer);
    fin.get(inputBuffer, 100, '\n'); fin >> ws ;
    maxValue = atoi(inputBuffer);

    cout << endl << "**********************" << endl;
    cout << label << endl;
    cout << "cellCount - " << cellCount << endl;
    cout << "itemCount - " << itemCount << endl;
    cout << "runCount - " << runCount << endl;
    cout << "maxValue - " << maxValue << endl;
    cout << "**********************" << endl;
    cout << "set\t";

    cells = new long[cellCount];
    for(int i=0; i< cellCount;i++)
    {
        cells[i] = 0;
    }
    lessThanX = new long[itemCount+1];
    for(int i=0; i< itemCount+1;i++)
    {
        lessThanX[i] = 0;
    }
```

```
for(int i=0; i<=maxValue; i++)
{
    cout << i << "\t";
}
cout << endl;
for(int j=0; j< setCount; j++)
{
    for(int run=1; run <= runCount; run++)
    {
        for(int i=0; i<itemCount; i++)
        {
            int cellNo = random(cellCount);
            cells[cellNo]++;
        }
        int maxVal = 0;
        for(int i=0; i<cellCount; i++)
        {
            if(cells[i] > maxVal)
            {
                maxVal = cells[i];
            }
        }
        for(int i=0; i<= maxVal; i++)
        {
            lessThanX[i]++;
        }
        for(int i=0; i<cellCount; i++)
        {
            cells[i] = 0;
        }
        cout << (j+1) << "\t";
        for(int i=0; i<=maxValue; i++)
        {
            cout << ((double)lessThanX[i])/runCount << "\t";
            lessThanX[i] = 0;
        }
        cout << endl;
    }
    delete[] lessThanX;
    delete[] cells;
}
fin.close();
return 0;
}
```

A0.4 Parameters for the simulations

Table A0.4.1 Parameters for the simulations

Label	Cell Count	Item Count	Run Count	Set Count	Max Value
Table 6.5.2 Project B Internal cells	56	82	50000	20	16
Table 6.5.2 Project B Row totals	8	82	50000	20	33
Table 6.5.2 Project B Column totals	7	82	50000	20	32
Table 6.5.2 Project C Internal cells	56	31	50000	20	10
Table 6.5.2 Project C Row totals	8	31	50000	20	20
Table 6.5.2 Project C Column totals	7	31	50000	20	19
Table 6.6.3 Project B Internal cells	40	83	50000	20	25
Table 6.6.3 Project B Row totals	5	83	50000	20	44
Table 6.6.3 Project B Column totals	8	83	50000	20	33
Table 6.6.3 Project C Internal cells	40	25	50000	20	9
Table 6.6.3 Project C Row totals	5	25	50000	20	14
Table 6.6.3 Project C Column totals	8	25	50000	20	20
Table 6.6.4 Project B Internal cells	35	92	50000	20	23
Table 6.6.4 Project B Row totals	5	92	50000	20	44
Table 6.6.4 Project B Column totals	7	92	50000	20	36
Table 6.6.4 Project C Internal cells	35	28	50000	20	12
Table 6.6.4 Project C Row totals	5	28	50000	20	14
Table 6.6.4 Project C Column totals	7	28	50000	20	19
Table 7.4.2 Project B Internal cells	20	51	50000	20	12
Table 7.4.2 Project B Row totals	4	51	50000	20	19
Table 7.4.2 Project B Column totals	5	51	50000	20	35
Table 7.4.2 Project B Poor progress	15	51	50000	20	20
Table 7.4.2 Project C Internal cells	20	71	50000	20	17
Table 7.4.2 Project C Row totals	4	71	50000	20	24
Table 7.4.2 Project C Column totals	5	71	50000	20	41
Table 7.4.2 Project C Poor progress	15	71	50000	20	25
Table 8.4.2 Project B Internal cells	36	56	50000	20	12
Table 8.4.2 Project B Row totals	6	56	50000	20	37
Table 8.4.2 Project B Column totals	6	56	50000	20	22
Table 8.4.2 Project C Internal cells	36	48	50000	20	16
Table 8.4.2 Project C Row totals	6	48	50000	20	17
Table 8.4.2 Project C Column totals	6	48	50000	20	38

Table A0.4.1 summarises the parameters that were used for each simulation. See Table A0.3.1 for an explanation of each parameter, and the order in which they were input into the program.

A0.5 Additional assumptions for the simulations

Three additional assumptions were made for the simulations:

1. A number of the tables include 'Unknown' categories. These categories were excluded from the calculations because it is not clear what information these categories communicate. For example, an Unknown category might 'hide' within it an additional process area.

2. References categorised as Unknown occur randomly within the evidence.

3. Classifications were typically generated separately for Projects B and C, and then merged. Because the classifications were generated separately for the two projects, categories existed in one classification that didn't exist in the other. For example, with Table 7.4.2, Build, Other projects, Early market support and Project management were not identified as a 'source' process areas for Project C. Although certain categories were not identified in the classification, typically these categories did exist for the two projects. So, again with Table 7.4.2, although Build was not identified as a process area it did actually exist with Project C.

Appendix B1 The selection of projects for case study

B1.1 Introduction

This appendix describes the generation of an initial set of candidate projects, the criteria for selecting cases from that candidate list, and the practical problems that reduced the initial set of five cases to a final set of two. This appendix complements chapter three.

B1.2 The selection of projects for case studies

Table B1.2.1 A summary of the candidate projects

Project	Purpose	Platform	Type of software	Status
A	New version	Mainframe	Global, real-time, middleware	Pre-plan
B	New version	Mainframe	Global, real-time, middleware	Just completed planning phase
C	Porting product	Desktop	Local, real-time, middleware	Just completed planning phase
D	New version	Mainframe	Message queueing	Project completed
E	Porting product	Mainframe	Message queueing	Post-plan
F	New product	Other	Internet	In planning phase
G	New version	Mainframe	Message queueing	In planning phase
H	New version	Other	Digital telephone services	Post-plan
I	New version	Other	Digital telephone services	Project completed
J	New version	Other	Message queueing	Post-plan
K	New version	Mainframe	Message queueing	Project completed
L	New version	Mainframe	Global, real-time, middleware	Project completed
M	New version	Desktop / Workstation	Global, real-time, middleware	Post-plan
N1	New version	Other	Digital telephone services	Project completed
N2	New version	Workstation	Digital telephone services	Project completed
O	New version	Workstation	Global, real-time, middleware	Post-plan

Table B1.2.1 provides summary descriptions of the candidate projects for the case studies. The descriptions of the projects include the purpose of the project (i.e. to produce a new product, a new version, or a port of the product to a new platform), the platform on which the product would operate (i.e. mainframe, workstation, desktop, other), the type

of product (i.e. middleware, message queueing, digital telephone services), and the status of the project at the time the project was considered as a candidate for a case study (i.e. pre-plan, in the planning phase, just completed the planning phase, post-plan, and project completed).

The term 'global' refers to a system that is designed to operate across an entire organisation, even where the organisation is distributed across a number of physical sites. By contrast, a 'local' system is designed to operate within one site, similar in concept to a local area network (LAN). Middleware systems are designed to operate 'between' the operating system and the applications. Thus, middleware relies on the operating system in order to function and provides additional functionality to applications which the operating system cannot (or does not) provide.

All projects were taken from IBM Hursley Park, and were chosen as candidates through informal discussions with a 'co-ordinator' at that laboratory. The informal criteria for candidate projects were that the projects should be currently occurring or have completed recently, and that the project leaders would be available (i.e. they were still at the laboratory) and likely to be willing for their projects to be studied.

Table B1.2.2 Criteria for selecting projects for case study

#	Criterion	Value of criterion
1	Select five projects for case studies	Satisfies the advice provided by Eisenhardt and Orlikowski (see chapter three)
2	Projects that were planned to complete within the next 12 months.	This would allow an exploration of whether the project actually completed according to the planned schedule.
3	Projects that had recently exited their plan phase.	Planning information would be easier to gather. Interviews on the plan would be closer to the plan phase, and thus might reveal more information. This would also allow investigations of some of the earlier processes in a project, such as high-level design. It might also reveal some information on 'plan-churn' and 'requirements-churn'.
4	Projects that progressed through more stages of the project.	Again, this would allow an investigation of more processes and more varied processes in the project.
5	Projects that were representative of the laboratory	There are various 'parameters' that one might consider. Table B1.2.1 presents some of these.
6	Projects that were not representative of the laboratory	

The heuristics presented in chapter three were used to establish criteria for selecting candidate projects as case studies. Table B1.2.2 summarises these criteria. Criteria 2, 3 and

4 imply that projects whose post-plan duration was 12 months were preferred. This naturally biases this investigation to shorter projects. For example, some projects at the laboratory are planned to take 18 months to two years, and in unusual circumstances longer. A perspective of shorter projects not only made the research feasible but also allowed an investigation of the life-cycle of a project.

Criteria 4 and 5 deliberately contradict each other. Clearly, representative projects are desireable because they support the development of a theory with wider applicability. Unrepresentative projects are also desireable because they define the limits of applicability. These two criteria are consistent with positions taken by Yin ([140]) and Eisenhardt ([39])

As explained in chapter three, five projects were initially selected for case studies (Projects A, B, C, E and F). It quickly became clear, however, that there would be difficulties with gaining frequent access to project members for Projects E and F. These two projects were dropped as case studies and Project G was introduced to compensate for the reduction in the number of case studies.

As the collection of evidence progressed, it became clear that it was extremely demanding to collect evidence for four projects (particularly when one considers attempts to interview project members regularly, and to record the details of these interviews). It was also clear that the analysis of the evidence (which was growing enormously) would also be extremely demanding. Consequently, the selection of case studies was further revised, reducing the number of projects to two (Projects B and C). Two case studies was considered to be the minimum number of acceptable projects, as two projects would allow cross-case comparison (see heuristic #2 in Table 3.3.1). Projects A and G were dropped because both projects were experiencing difficulties in completing their plan phases. (In addition, Project A was not originally planned to complete within the 12 month time-frame for the data collection.) The problems that these two projects were experiencing would be very interesting with regards to project schedule. The delay in completing the plan phases, however, meant that less project time would be spent on other project phases within the time-frame set for the evidence collection. Other researchers have documented the problems they experienced with long, intensive, qualitative case studies (i.e. [78])

Table B1.2.3 A summary of the final status of the candidate projects

Status	Cases	Number of cases
Main cases with data collected and analysed	B & C	2
Main cases with data collected but not analysed	A & G	2
Secondary cases with data collected but not analysed	H, K & M	3
Secondary cases with no data collected but data available	I, L, N1 & N2	4
Cases deferred	D, E, F & J	4
Cases where project manager did not respond	O	1
Total		16

Table B1.2.3 summarises the status of the candidate projects at the time the evidence-collection period was completed. Two projects (Projects B and C) remain as main case studies, with the opportunity for analysis of evidence from a further five case studies (projects A, G, H, K, M). Due to practical constraints, the additional cases were not analysed.

Appendix B2 An explanation of the figures used in chapter five

B2.1 Introduction

Given the complexity of the figures presented in chapter five, some explanation of their structure and notation is appropriate. This appendix provides an explanation of the purpose, structure and notation used in the figures.

B2.2 Explanation of the purpose, structure and notation of the diagrams

The main purpose of the figures was to communicate a large volume of evidence from a variety of sources in the most efficient way. Miles and Huberman ([79]) suggest numerous techniques for organising qualitative evidence. One set of their techniques concerns organising multiple sources and types of evidence according to time. Also, the visualisation of evidence from a number of different sources (by placing that evidence within the same figure) may reveal subtle relationships between aspects of the project ([124, 125]). The visualisation of evidence in chapters five through eight permits comparisons of different types of evidence across those chapters e.g. comparison of the frequencies of waiting presented in chapter six with the project-level behaviour presented in chapter five.

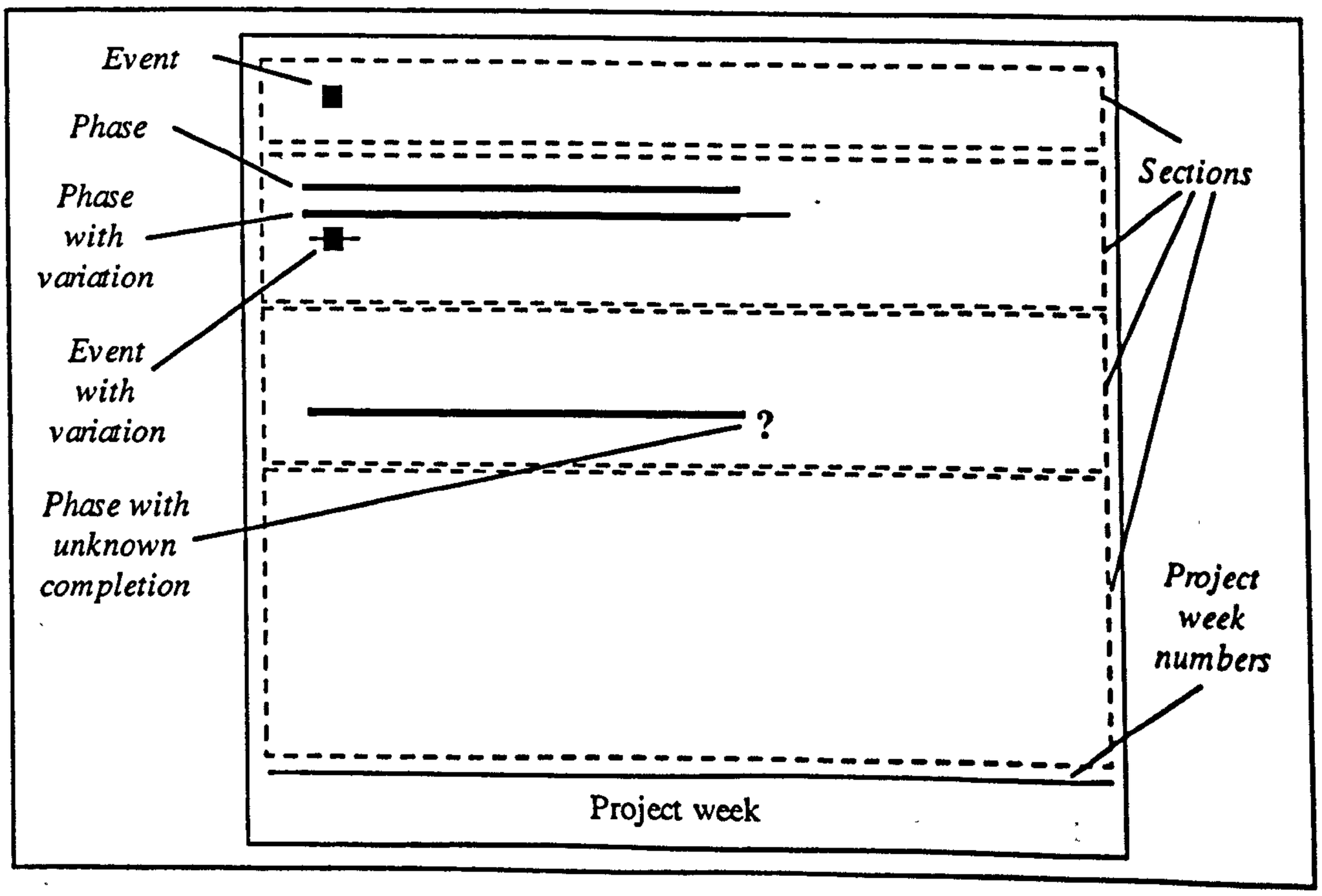


Figure B2.2.1 A simplified example of a figure used in chapter five

Figure B2.2.1 presents a simplified example of one of the figures used in chapter five. The figure indicates the following:

- All information presented in the diagrams is positioned according to a project week number. Project week numbers are identified on the x axis of every diagram.
- The vertical double-line at week 52 for Project B and weeks 48 and 59 for Project C indicate the planned delivery of the product.
- Each diagram is vertically organised into a number of sections, with each section containing information of a certain type (e.g. schedule information, defect information).
- With graphs, the y-axis is used to represent a quantity of some unit. For example, the y-axis on the Defect section on Figure 5.3.5 refers to the number of defects in the Open, Accepted and Answered states.
- Small black squares represent the occurrence of some event. An event might be a planned or actual milestone (also known as a Decision Checkpoint), or a reference to something that has happened in the project. For example, in Figure 5.3.1, the black square, at week 18, with the text 'DCR cutoff' is an event. In the re-plans sections, small black squares represent the occurrence of an internal milestone and small black circles represent the occurrence of an external milestone.
- Long, horizontal thin black bars represent some planned variation in the start or completion of a phase or the occurrence of a milestone. For example, in Figure 5.3.1 the planning phase of Project B was planned to complete between week -3 and week -1. Concomitant with the planned completion of the plan phase, the planned occurrence of the Plan DCP had a planned variation between week -3 and -1.
- Long, horizontal medium-sized black bars represent the planned or actual occurrence of a phase. For example, in Figure 5.3.1 the planned design phase lasts from week 1 through week 19.
- Long, horizontal *broken* medium-sized bars indicates that a phase has formally completed, but work relating to the phase continues.
- A question mark is used to indicate some doubt as to the exact week that some event occurred or some phase started or completed. For example, there is doubt as to when the 6th re-plan occurred for Project B (see Figure 5.3.2).
- Particular features are identified through abbreviations. For example, the third feature on Project B is identified as F03 (see Figure 5.3.2).
- A number of acronyms are used in the diagrams: DCP refers to a decision checkpoint; FV refers to functional verification; OO refers to object-oriented; ST refers to system test; Y2K refers to year-2000.

Appendix B3 Evidence on project behaviour

B3.1 Introduction

This appendix provides detailed evidence to support the evidence and analyses presented and discussed in chapter five. Section B3.2 provides details on Project B. Section B3.3 provides details on Project C. For reasons of confidentiality, certain information is either disguised or not disclosed.

B3.2 Evidence from Project B

Table B3.2.1 Resource breakdown per month

Month	Week	Staff-level
Aug Year 1	-12 through -9	28
Sep	-8 through -4	37
Oct	-3 through 1	38.5
Nov	2 through 5	40.5
Dec	6 through 9	44
Jan Year 2	10 through 14	52.5
Feb	15 through 18	52.5
Mar	19 through 22	52.5
Apr	23 through 26	57.5
May	27 through 31	54.5
Jun	32 through 35	56.5
Jul	36 through 39	60.5
Aug	40 through 44	54.5
Sep	45 through 48	31
Oct	49 through 53	29
Nov	54 through 57	24
Total		713

Table B3.2.2 Resource breakdown per process area

Resource area	Year 1 < Week 10	Year 2 Week 10+	Total	%
Management	17	46	63	8.00
Product B Development	86.5	146	232.5	29.5
Product BS Development	0	38	38	4.8
'Dots' Development	10	14	24	3.1
System Test	20	131	151	19.2
InformationDevelopment	35	81	116	14.7
Performance	9	23	32	4.1
Support Management	9	30	39	4.9
Build	14	28	42	5.3
Library Control System	18	32	50	6.4
Total	218.5	569	787.5	100

Note:

1. 'Dots' development refers to the work required to update version numbers etc.
2. 'Product BS' refers to a companion product that was 'bundled' with Project B

Table B3.2.3 The planned occurrence of key project milestones

Milestone/Phase	Planned weeks
Concept decision checkpoint	-12 through -9
Plan decision checkpoint	-4 through 1
Design, code & unit test complete	19 through 23
Availability decision checkpoint	37 through 40
System test complete	41 through 44
Announce decision checkpoint	45 through 49
Integration test complete	50 through 53
General availability of the product	50 through 53

Table B3.2.4 The planned occurrence of project milestones at week 33

Milestone	Planned week
Design, code and unit test complete	31
Functional verification complete	36
Announce	46
General availability	52

Table B3.2.5 The actual occurrence of key project milestones

Milestone	Actual week
System test complete	58
'Ship' to manufacturing	49
General availability of the product	52

Table B3.2.6 The original plan and the subsequent re-plans

Plan / Re-plan	Week		Contents of re-plan
1st Plan	-2	1.	Original plan accepted at the Plan Decision Checkpoint
1st re-plan	20	1.	F03's functional verification completion revised to week 36.
		2.	Caused by slow progress of F03 design/code phase
2nd re-plan	24	1.	Rewriting test plan to move F07 testing to back of plan.
		2.	Caused by slow progress on testing F07.
3rd re-plan	27	1.	F02's functional verification completion revised to week 34.
		2.	Caused by slippage in F02 design/code phase.
4th re-plan?	35	1.	Recognised that F03's functional verification will not complete week 36 due to delay in design/code phase.
5th re-plan	42	1.	F02 team feel that they can complete functional verification testing by week 44.
6th re-plan	??	1.	F02's System Test completion revised to week 47
7th re-plan	46	1.	F02's System Test completion revised to week 49
		2.	F03's System Test completion revised from week 49

Table B3.2.7 Events concerning the increase in activity

Planned events	Week of occurrence
Building of weekly increments starts	22

Unplanned events	Week of occurrence
Testers are working shifts and weekends	27
Designer starts "mission pay"	37
Daily defect Screen Team meetings start	38
Major concerns re project schedule ...	39 through 41
... with decision to retain current schedule	41
Designer prioritising defects so that it is clear which defects must be fixed.	41
Request for twice-weekly builds.	42
Categorising defects to determine which ones can be passed to Project B+1, and which ones should be prioritised.	43
Push to get as many defects through in order to make manufacturing build	45
Development meet with system test to determine any other specific areas that are stopping test.	49

Table B3.2.8 Summary of functional verification status

%A = Percentage attempted %F = Percentage failed

	Feature							
	F02		F03		F07		F09	
Week	%A	%F	%A	%F	%A	%F	%A	%F
-3 to 28								
29	53	4	45	1	100	4		
30	58	5	50	1	100	4	0	0
31								
32	58	5	66	3	100	4	0	0
33								
34	58	5	75	3	100	4	0	0
35	58	5	77	3	100	4		
36	72	5	80	2	100	0	0	0
37	72	5	85	5	100	0		
38	72	5	86	5	100	0		
39	72	5	87	4	100	0		
40	72	5	87	4	100	0		
41	80	7	94	3	100	0	100	0
42	77	5	95	3	100	0	100	0
43	80	6	96	3	100	0	100	0
44	80	4	98	3	100	0	100	0
45	88	6	99	3	100	0	100	0
46	88	6	99	3	100	0	100	0
47	87	5	99	2	100	0	100	0
48	87	5	99	2	100	0	100	0
49	87	3	99	1	100	0	100	0
50	87	3	99	1	100	0	100	0
51	89	4	99	1	100	0	100	0
52	91	4	99	1	100	0	100	0
53								
54	92	4	99	1	100	0	100	0
55								
56	99	2	99	1	100	0	100	0
57								
58	99	2	99	1	100	0	100	0
59								

B3.3 Evidence from Project C

Table B3.3.1 Feature status

Week	Total	Open	Design	Size	Commit	Complete
3	19		8	8	3	
5	19		8	5	6	
8	19		7	5	7	
11	19		5	7	7	
13	20		4	9	7	
15	20		4	9	7	
19	21		3	8	10	
21	21		2	9	10	
24	24	1	2	8	12	1
25	23		2	8	12	1
26	23		2	8	12	1
27	23		2	8	12	1
28	23		2	8	12	1
29	23		2	8	12	1
30	27		3	6	17	1
32	27		3	3	19	2
33	30		6	2	20	2

Table B3.3.2 Product sizings (thousands of lines of code)

	KLOC				
Platform	**Re-used**	**New**	**Changed**	**Ported**	**Total**
DOS	245		5		250
OS/2	225	3	2		230
New O.S.		15		55	70
Common	148	1	1		150
Total	618	19	8	55	700

Table B3.3.3 Resource plan

Process area	1Q	2Q	3Q	4Q
Support	12	12	12	13
Level 3	3	3	3	3
New product (O.S.) development				
Development	3	3	2	
Information Planning	1	1	1	1
System Test			2	2
Total development	19	21	20	16
Total 'development ' for year:	19 person years.			
Marketing	1	1	1	1
Technical Planning	1	1		
Product Planning	0.5	0.5	0.5	

Table B3.3.4 The original plan and the subsequent re-plans

Plan / Re-plan	Week	Comments
1st Plan	3 & 8	1. Original plan accepted at the Plan DCPs
1st Re-plan	22	1. Re-plan following Interim Commit Checkpoint
		2. Extend the duration of the DCUT phase
		3. Staged-entry into Acceptance Test and System Test
2nd Re-plan	39	1. Formal plan change, accepted by senior management
		2. Introduction of new Year-2000 requirements
		3. Introduction of JAVA feature
		4. Re-schedule the product delivery date
3rd Re-plan	51	1. Extend the duration of the System Test phase
		2. Compress the Manufacturing phase
		3. Compress the Gold Code Production phase
4th Re-plan	54	1. Extend the duration of the System Test phase
		2. Compress the Manufacturing phase
		3. Compress the Gold Code Production phase

Appendix B4 The waiting evidence

B4.1 Introduction

This appendix provides detailed evidence to support the analysis presented in chapter six. The following information is included:

- The classification of each item of waiting evidence.
- The results of the Mann Whitney *U* tests of the prevalence for waiting during the end of the project rather than during the middle of the project, for the two projects.
- Adjustments to Bradac *et al.*'s ([18]) data.

B4.2 The classifications of items of waiting evidence

Tables B4.2.1 and B4.2.2 present the 'raw' waiting evidence for Projects B and C respectively. Each item is labelled with the week in which it occurred in the status meetings. Due to the sensitivity of the information, the entire text for each reference cannot be presented.

Table B4.2.1 Waiting evidence for Project B

Week	Phrase	Initial classification	Source process area	Dependent process area	Refined classification	Bradac *et al.*'s classification
3	hold	Plan churn	Project management	Design/Code	Other	Other
12	awaiting	Code	Design/Code	Design/Code	Code	Software
13	waiting	Features	Other project	Design/Code	Other	Other
14	stoppers	Unknown	Not Applicable	Performance	Unknown	Other
15	waiting	Technology	Organisation	Performance	Other	Hardware
16	waiting	Code	Other project	Performance	Code	Software
18	waiting	Discussion	Design/Code	Design/Code	Decision	Other
20	stopper	Organisation	Other project	Project management	Other	Other
20	awaiting	Decision	Organisation process	Project management	Decision	Other
21	waiting	Code	Build	System Test	Code	Software
21	holding up	Defect	Unknown	Build	Defect/Fix	Software
21	stopping	Defect	Unknown	Build	Defect/Fix	Software
21	awaiting	Discussion	Other project	Project management	Decision	Other
21	waiting	Decision	Organisation process	Project management	Decision	Other
21	waiting	Code	Build	System Test	Code	Software
22	awaiting	Discussion	Other project	Project management	Decision	Other
22	still	Decision	Organisation process	Project management	Decision	Other
22	waiting	Fix	Unknown	Other project	Defect/Fix	Software
22	waiting	Information	Design/Code	Other project	Information	Expert
22	awaiting	Resource	Unknown	Unknown	Resource	Other
24	awaiting	Discussion	Other project	Project management	Decision	Other
24	waiting	Decision	Organisation process	Project management	Decision	Other
25	waiting	Decision	Organisation process	Information Development	Decision	Other
25	waiting	Decision	Organisation process	Unknown	Decision	Other
25	waiting	Decision	Unknown	Early Market Support	Decision	Other
26	waiting	Decision	Organisation process	Unknown	Decision	Other
26	waiting	Decision	Organisation	Project management	Decision	Other
26	awaiting	Decision	Organisation process	Information development	Decision	Other
27	waiting	Decision	Organisation	Project management	Decision	Other

Table B4.2.1 Waiting evidence for Project B

Week	Phrase	Initial classification	Source process area	Dependent process area	Refined classification	Bradac *et al.*'s classification
27	waiting	Decision	Organisation	Project management	Decision	Other
27	awaiting	Decision	Organisation process	Information development	Decision	Other
29	stopping	Unknown	Unknown	Design/Code	Unknown	Other
29	waiting	Decision	Organisation process	Project management	Decision	Other
29	awaiting	Decision	Organisation process	Information development	Decision	Other
32	waiting	Decision	Other project	Performance	Decision	Other
32	waiting	Decision	Other project	Design/Code	Decision	Other
32	waiting	Decision	Organisation process	Project management	Decision	Other
32	awaiting	Decision	Organisation process	Information development	Decision	Other
33	stopping	Problem	Other project	System Test	Defect/Fix	Software
33	waiting	Decision	Other project	Design/Code	Decision	Other
33	waiting	Decision	Organisation process	Project management	Decision	Other
33	awaiting	Decision	Organisation process	Information development	Decision	Other
33	awaiting	Decision	Other project	Project management	Decision	Other
34	awaiting	Code	Design/Code	Test	Code	Software
34	waiting	Code	Other project	Test	Code	Software
34	waiting	Decision	Other project	Design/Code	Decision	Other
34	waiting	Decision	Organisation process	Project management	Decision	Other
34	awaiting	Decision	Organisation process	Information development	Decision	Other
34	waiting	Decision	Other project	Project management	Decision	Other
35	waiting	Code	Design/Code	Test	Code	Software
35	waiting.	Decision	Other project	Design/Code	Decision	Other
35	waiting	Decision	Organisation process	Project management	Decision	Other
36	holding up	Defect	Design/Code	Design/Code	Defect/Fix	Software
36	waiting	Decision	Organisation process	Project management	Decision	Other
37	waiting	Decision	Other project	Design/Code	Decision	Other
37	waiting	Code	Design/Code	Build	Code	Software
37	waiting	Resource	Not Applicable	Test	Resource	Other
37	waiting	Decision	Organisation process	Project management	Decision	Other
38	awaiting	Code	Design/Code	Test	Code	Software

Table B4.2.1 Waiting evidence for Project B

Week	Phrase	Initial classification	Source process area	Dependent process area	Refined classification	Bradac *et al.*'s classification
38	waiting	Fix	Other project	Test	Defect/Fix	Software
38	waiting	Code	Other project	Early Market Support	Code	Software
39	held up	Defect	Other project	Test	Defect/Fix	Software
39	waiting	Fix	Other project	Build	Defect/Fix	Software
39	waiting	Code	Design/Code	Test	Code	Software
39	waiting	Code	Build	Test	Code	Software
40	waiting	Fix	Other project	Test	Defect/Fix	Software
40	waiting	Fix	Other project	Unknown	Defect/Fix	Software
40	waiting	Resource	Not Applicable	Test	Resource	Other
40	waiting	Decision	Other project	Project management	Decision	Other
40	waiting	Fix	Other project	Performance	Defect/Fix	Software
40	waiting	Fix	Other project	Performance	Defect/Fix	Software
41	holding up	Defect	Other project	Test	Defect/Fix	Software
41	stoppers	Problem	Other project	Design/Code	Defect/Fix	Software
42	holding up	Defect	Other project	Test	Defect/Fix	Software
42	stopping	Defect	Unknown	Test	Defect/Fix	Software
43	stopped	Change to environment	Build	Design/Code	Other	Other
44	stopping	Defect	Not Applicable	Test	Defect/Fix	Software
44	waiting	Fix	Other project	Performance	Defect/Fix	Software
44	waiting	Fix	Other project	Performance	Defect/Fix	Software
44	waiting	Fix	Other project	Unknown	Defect/Fix	Software
44	awaiting	Decision	Organisation process	Project management	Decision	Other
44	waiting	Decision	Unknown	Unknown	Decision	Other
45	waiting	Code	Build	Test	Code	Software
45	awaiting	Fix	Design/Code	Test	Defect/Fix	Software
45	awaiting	Decision	Organisation process	Project management	Decision	Other
46	waiting	Code	Build	Test	Code	Software
46	waiting	Fix	Other project	Performance	Defect/Fix	Software
46	awaiting	Material	Design/Code	Information development	Information	Documentation
46	waiting	Fix	Design/Code	Test	Defect/Fix	Software

Table B4.2.1 Waiting evidence for Project B

Week	Phrase	Initial classification	Source process area	Dependent process area	Refined classification	Bradac *et al.*'s classification
46	waiting	Decision	Other project	Project management	Decision	Other
47	stopped	Problem	Unknown	Performance	Defect/Fix	Software
47	waiting	Material	Design/Code	Information development	Information	Documentation
47	waiting	Material	Design/Code	Information development	Information	Documentation
48	waiting	Fix	Design/Code	Test	Defect/Fix	Software
49	waiting	Code	Design/Code	Performance	Code	Software
49	waiting	Code	Design/Code	Performance	Code	Software
49	waiting	Authorisation	Unknown	Build	Decision	Other
49	awaiting	Decision	Other project	Build	Decision	Other
50	waiting	Code	External organisation	Early Market Support	Code	Software
51	waiting	Fix	Design/Code	Test	Defect/Fix	Software
52	waiting	Decision	Other project	Design/Code	Decision	Other
56	waiting	Fix	Design/Code	Performance	Defect/Fix	Software
56	waiting	Code	Unknown	Build	Code	Software
58	stopped	Defect	Build	Build	Defect/Fix	Software

Table B4.2.2 Waiting evidence for Project C

Week	Phrase	Source Process Area	Dependent Process Area	Initial classification	Refined classification	Bradac *et al.*'s classification
11	waiting	Not Applicable	Not Applicable	Not Applicable	Not Applicable	Other
11	waiting	Development	Development	Code	Code	Software
19	awaiting	Marketing	Development	Response	Decision	Other
21	on hold	Development	Development	Redesign	Other	Software
24	awaiting	Development	Development	Fix	Defect/Fix	Software
26	awaiting	Organisation process	System Test	Technology	Other	Hardware
27	awaiting	Marketing	System Test	Information	Information	Expert
32	awaiting	Unknown	Unknown	Decision	Decision	Other
33	blocking	Development	System Test	Defect	Defect/Fix	Software
33	hold off.	Development	Development	Quality improvement	Other	Other
34	blocking	Development	Acceptance Test	Defect	Defect/Fix	Software
35	awaiting	Not Applicable	Not Applicable	Not Applicable	Not Applicable	Other
35	awaiting	Development	Development	Code	Code	Software
35	awaiting	Development	Development	Approval	Decision	Other
35	blocking	Development	Acceptance Test	Lack of functionality	Other	Other
36	blocking	Unknown	System Test	Testcase	Other	Other
42	blocking	Development	Longevity Test	Unstable product	Other	Software
43	blocked	Unknown	System Test	Problem	Defect/Fix	Software
43	on hold	Unknown	System Test	Service commitments	Other	Other
45	waiting	Development	Information Development	Review comments	Information	Review
45	awaiting	Development	Information Development	Initial input	Information	Other
45	waiting	Unknown	Unknown	Unknown	Other	Other
46	awaiting	Development	Information Development	Review input	Information	Review
46	blocking	Unknown	System Test	Problem	Defect/Fix	Software
46	on hold	System Test	System Test	Vacation	Other	Other
47	awaited	Development	Information Development	Code	Code	Software
47	waiting	Unknown	Unknown	Unknown	Other	Other
47	blocking	Development	System Test	Defect	Defect/Fix	Software
47	blocking	Development	System Test	Problem	Defect/Fix	Software

Table B4.2.2 Waiting evidence for Project C

Week	Phrase	Source Process Area	Dependent Process Area	Initial classification	Refined classification	Bradac *et al.*'s classification
48	blocked	Development	System Test	Defect	Defect/Fix	Software
49	awaiting	External organisation	Development	Approval	Decision	Other
49	awaiting	Not Applicable	Not Applicable	Not Applicable	Not Applicable	Other
49	awaiting	Development	System Test	Fix verification	Defect/Fix	Software
50	awaiting	Development	System Test	Fix verification	Defect/Fix	Software
51	blocking	Development	System Test	Defect	Defect/Fix	Software
51	blocking	Unknown	System Test	Defect	Defect/Fix	Software
52	awaiting	Other project	Development	Response	Decision	Other
52	awaited	Other project	Development	Response	Decision	Other
52	blocking	Other project	System Test	Defect	Defect/Fix	Software
53	awaiting	External organisation	Development	Response	Decision	Other
53	awaiting	Other project	Development	Information	Information	Expert
57	awaiting	Build	Development	Fix	Defect/Fix	Software

B4.3 The Mann Whitney U tests of the prevalence of waiting

Presented below are the results of the Mann Whitney *U* tests of the prevalence for waiting during the end of the project rather than during the middle of the project, for Projects B and C. The results are taken directly from *Data Desk* v6.0, a data analysis software package for the Apple Macintosh.

Mann Whitney *U* test for Project B

Ho: Median1 = Median2 Ha: Median1 > Median2
Individual Alpha Level 0.00100
Ties Included

End:Yes - Middle:Yes :
Test Ho: Median(End:Yes) = Median(Middle:Yes) vs Ha: Median(End:Yes) > Median(Middle:Yes)

	Rank Totals	Cases	Mean Rank
End:Yes	818	29	28.19
Middle:Yes	264	17	15.50
Total	1760	46	38.26
Ties Between Groups	679	34	19.97

U-Statistic: 382
U-prime: 110
Sets of ties between all included observations: 7
Variance: 1930.9
Adjustment To Variance For Ties: -65.376
Expected Value: 246.50
z-Statistic: 3.1487
p = 0.0008
Reject Ho at Alpha = 0.00100

Mann Whitney *U* test for Project C

Individual Alpha Level 0.0100
Ho: Median1 = Median2 Ha: Median1 > Median2
Ties Included

End:Yes - Middle:Yes :
Test Ho: Median(End:Yes) = Median(Middle:Yes) vs Ha: Median(End:Yes) > Median(Middle:Yes)

	Rank Totals	Cases	Mean Rank
End:Yes	415	20	20.75
Middle:Yes	146	13	11.23
Total	939	33	28.45
Ties Between Groups	378	27	14

U-Statistic: 205
U-prime: 55
Sets of ties between all included observations: 5
Variance: 736.67
Adjustment To Variance For Ties: -59.337
Expected Value: 130
z-Statistic: 2.8818
p = 0.0020
Reject Ho at Alpha = 0.0100

B4.4 Adjustments to Bradac *et al.*'s data

As explained in chapter five, the evidence collected from Projects B and C only include references to waiting, whereas Bradac *et al.* included references to working the process as well as references to waiting on blocked work. Bradac *et al.*'s percentages of waiting were adjusted in chapter five, for the purposes of comparison, by removing the influence of working the process.

Table B4.4.1 Comparison of types of waiting with Bradac *et al.*'s classification

	Project B		Bradac *et al.*		Project C	
Category	**Count**	**% total**	**% waiting**	**% time**	**Count**	**% total**
Other	53	51.4	66.7	40.7	18	42.9
Review	0	0	15.1	9.2	2	4.8
Expert	1	1.0	5.1	3.1	2	4.8
Laboratory	0	0	4.5	2.7	0	0
Documentation	3	2.9	3.9	2.4	0	0
Software	45	43.7	3.1	1.9	19	45.2
Hardware	1	1.0	1.6	1	1	2.3
Total	103	100	100	61	42	100

Table B4.4.1 presents a more detailed version of Table 3.7.2. Bradac *et al.*'s original percentages are shown in the '% time' column. This indicates that of all the different states that the lead engineer was in, 61% were spent waiting and, by implication, 39% were spent working the process. *Within* the waiting states, however, waiting on other accounted for about 67% of the waiting. In the current investigation, there was no firm foundation for establishing the ratio of waiting to working the process, so the investigation concentrated on the breakdown of waiting only.

Appendix B5 The progress of work evidence

B5.1 Introduction

This appendix provides detailed evidence to support the analysis presented in chapter seven. The following information is included:

- The classification of each item of progress of work evidence.
- The results of the Mann Whitney *U* tests of the prevalence for poor progress during the end of the project rather than during the middle of the project, for the two projects.

B5.2 The classifications of items of progress evidence

Tables B5.2.1 and B5.2.2 present the 'raw' progress of work evidence for Projects B and C respectively. Each item is labelled with the week in which it occurred in the status meetings. Due to the sensitivity of the information, the entire text for each reference cannot be presented.

Table B5.2.1 Progress of work evidence for Project B

Week	Phrase	Type of progress	Process Area	Work causing poor progress
14	in progress	Making progress	Unknown	Unknown
19	trying to make progress	Slow progress	FV testing	Defect/Fix
19	no progress	No progress	Performance testing	Defect/Fix
20	slow progress	Slow progress	Testing	Unknown
21	no real progress	No progress	Testing	System reliability problems
21	no real progress	No progress	Performance testing	System reliability problems
25	reasonable progress	Reasonable progress	Testing	System reliability problems
25	turnaround slow	Slow progress	Unknown	Unknown
25	not able to make any real progress	No progress	Performance testing	Defect/Fix
26	progress... slow	Slow progress	Testing	System reliability problems
32	progressing well	Good progress	FV testing	Not applicable
32	in progress	Making progress	System test	Unknown
32	in progress	Making progress	System Test	Unknown
35	no real progress	No progress	Unknown	Unknown
36	holding up progress	No progress	Unknown	Defect/Fix
36	progress has been slow	Slow progress	Unknown	System reliability problems
36	good progress	Good progress	Performance testing	Not Applicable
36	progressing	Reasonable progress	ID	Unknown
36	making steady progress	Reasonable progress	System testing	Not Applicable
36	no progress	No progress	System testing	Absence
37	little progress	No progress	Testing	Sickness
37	good progress	Good progress	Unknown	Not Applicable
37	making steady progress	Reasonable progress	Testing	Not Applicable
38	slow progress	Slow progress	Testing	Defect/Fix
38	making progress	Making progress	Unknown	Unknown
38	slow progress	Slow progress	Unknown	Defect/Fix
38	progressing	Making progress	Testing	Unknown
39	progress is very slow	Slow progress	Testing	Unknown
39	progress is not being made	Slow progress	Unknown	Defect/Fix

Table B5.2.1 Progress of work evidence for Project B

Week	Phrase	Type of progress	Process Area	Work causing poor progress
	as quickly as we would like			
40	little progress	No progress	Development	Unknown
40	unable to make desired progress	Slow progress	Unknown	System reliability problems
41	making progress	Making progress	System testing	Unknown
41	progress that stress hoped to make wasn't achieved	Slow progress	Stress testing	System reliability problems
42	steady progress	Reasonable progress	FV testing	Not Applicable
42	good progress	Good progress	FV testing	Not Applicable
42	now progressed	Making progress	Unknown	Defect/Fix
42	make quicker progress.	Better progress	Testing	Not Applicable
42	progressing	Making progress	Testing	Unknown
43	progress is now being made	Making progress	FV testing	Not Applicable
43	still making progress	Slow progress	Stress testing	Unknown
43	progressing slowly	Slow progress	Unknown	Unknown
44	made some progress	Slow progress	Testing	Unknown
44	slow progress	Slow progress	Testing	Unknown
45	impacting progress	Slow progress	Development	Prep of Skills Transfer
45	progress is good	Good progress	FV testing	Not Applicable
45	did make progress	Making progress	System testing	Unknown
45	made a lot of progress	Good progress	System testing	Not Applicable
46	being progressed.	Making progress	System testing	Unknown
46	no progress	No progress	Development	MPUs
46	making progress	Making progress	System testing	Not Applicable
46	progressing well	Good progress	Development	Not Applicable
47	no real progress	No progress	Development	Prep of Skills Transfer
47	good progress	Good progress	System testing	Defect/Fix
47	progress slow	Slow progress	System testing	Defect/Fix
47	not made much progress	No progress	Stress testing	Defect/Fix
48	not making the progress expected	Slow progress	System testing	Unknown

Table B5.2.1 Progress of work evidence for Project B

Week	Phrase	Type of progress	Process Area	Work causing poor progress
48	progressing	Making progress	ID	Not Applicable
48	still not making any progress	No progress	System testing	Unknown
48	progressing well	Good progress	System testing	Defect/Fix
49	enabling FV to progress	Enabling progress	FV testing	Not Applicable
49	better progress	Better progress	Development	Unknown
50	making good progress	Good progress	Build	Not Applicable
50	now making good progress	Good progress	Testing	Not Applicable
51	making reasonable progress	Reasonable progress	Testing	Not Applicable
51	making reasonable progress	Reasonable progress	Testing	Not Applicable
51	progressing reasonably well	Reasonable progress	Testing	Not Applicable
52	making some progress	Slow progress	Testing	Defect/Fix

Table B5.2.2 Progress of work evidence for Project C

Week	Phrase	Initial classification	Process Area
3	Significant movement is now required	significant progress required	Development
3	progress	significant progress required	Development
3	impacting ability to progress	progress impacted	Development
5	Significant movement is required	significant progress required	Development
11	... Significant movement is still required	significant progress required	Development
11	progress... impacted	progress impacted	Development
13	work is urgently required to complete the design and to progress	progress required	Development
13	progress ... inhibited	no progress	Development
13	progress... slow	slow progress	Development
13	progressing well.	good progress	Development
15	progressing well	good progress	Development
19	progressed well	good progress	Development
21	progress was <10%	no progress	Development

Table B5.2.2 Progress of work evidence for Project C

Week	Phrase	Initial classification	Process Area
25	making progress	making progress	Development
25	little time to enable progress	no progress	Development
25	work is progressing	making progress	System Test
25	progress is on target	reasonable progress	System Test
26	still progressing	making progress	Development
26	progressing	making progress	System Test
26	unable to make ... progress	no progress	Development
26	little ... progress	no progress	Development
28	in progress	making progress	Testing
29	progressing well.	good progress	Development
29	progress... going to plan.	reasonable progress	Acceptance Test
29	strong progress	good progress	Development
30	progressing	making progress	Development
30	progressing	making progress	Development
30	progress has received an unplanned adverse impact	progress impacted	Development
30	progress ... interrupted	progress impacted	Development
30	strong progress	good progress	Development
30	progressing	making progress	Testing
32	progress **URGENT**	significant progress required	Development
32	progressing	making progress	Development
32	unable to progress	no progress	Development
32	only slow progress	slow progress	Development
33	progress **URGENT**	significant progress required	Development
33	made useful progress	making progress	Development
33	slow progress	slow progress	Development
33	good progress	good progress	Development
33	progressing well	good progress	System Test
34	progressing positively	good progress	Development
34	progress... impacted	progress impacted	Development
35	not made progress	no progress	Development

Table B5.2.2 Progress of work evidence for Project C

Week	Phrase	Initial classification	Process Area
35	before progressing further	no progress	Unknown
35	a further full week on progressing	no progress	Unknown
35	still in progress....	making progress	Unknown
36	able to progress	making progress	Development
38	slow improvement	slow progress	System Test
38	good progress	good progress	System Test
38	impact progress.	progress impacted	System Test
38	made progress	making progress	Development
38	progress	making progress	Unknown
40	slow progress	slow progress	Testing
40	good progress.	good progress	Testing
40	some progress	slow progress	Development
40	progress	making progress	Development
40	progress	making progress	Testing
40	progressing	making progress	Development
42	solid ... progress	good progress	Development
42	progress	making progress	Testing
42	progressing,.	making progress	System Test
42	progressing fast	good progress	Development
42	not progressed	no progress	Development
43	progress ... impacted	progress impacted	Development
43	progress... blocked	no progress	System Test
43	slow progress	slow progress	Unknown
43	in progress	making progress	Development
43	considerable progress	good progress	Development
45	constrained ...progress	slow progress	System Test
45	some progress had been made	slow progress	Unknown
46	blocking... progress.	no progress	System Test
46	progress... on hold...	no progress	System Test
46	no ... work in progress	no progress	System Test
47	significant progress	good progress	Service

Table B5.2.2 Progress of work evidence for Project C

Week	Phrase	Initial classification	Process Area
47	significant progress	good progress	Development
47	no progress	no progress	System Test
47	progress	making progress	System Test
47	progress	making progress	Testing
47	blocking progress	no progress	Unknown
48	good progress	good progress	System Test
48	strong ...progress	good progress	Development
48	little ... progress	no progress	System Test
48	... very little progress	no progress	System Test
48	positive progress	good progress	Development
49	encouraging progress	encouraging progress	Development
49	little effective progress	no progress	System Test
50	significant progress	good progress	System Test
51	good progress	good progress	Development
51	progress	making progress	System Test
51	holding up progress	no progress	System Test
51	inhibited progress	no progress	Testing
51	progressed	making progress	Testing
51	significant progress	good progress	Unknown
52	progress	making progress	Unknown
52	unable to progress	no progress	Unknown
52	progress	making progress	Testing
53	progress continues	making progress	Unknown
53	strong progress	good progress	Development
53	positive progress	good progress	Testing

B5.3 The Mann Whitney *U* tests of the prevalence for outstanding work

Presented below are the results of the Mann Whitney *U* tests of the prevalence for outstanding work during the end of the project rather than during the middle of the project, for Projects B and C. The results are taken directly from *Data Desk* v6.0, a data analysis software package for the Apple Macintosh.

Mann Whitney *U* test for Project B

Individual Alpha Level 0.01
Ho: Median1 = Median2 Ha: Median1 > Median2
Ties Included

End:No+Slow progress - Middle:No+Slow progress :
Test Ho: Median(End:No+Slow progress) = Median(Middle:No+Slow progress) vs Ha: Median(End:No+Slow progress) > Median(Middle:No+Slow progress)

	Rank Totals	Cases	Mean Rank
End:No+Slow progress	773.50000	29	26.67
Middle:No+Slow progress	307.50000	17	18.09
Total	2071	46	45.02
Ties Between Groups	990	44	22.50

U-Statistic: 338
U-prime: 154
Sets of ties between all included observations: 4
Variance: 1930.9
Adjustment To Variance For Ties: -414.52
Expected Value: 246.50
z-Statistic: 2.3626
p = 0.0091
Reject Ho at Alpha = 0.01

Mann Whitney *U* test for Project C

Individual Alpha Level 0.05
Ho: Median1 = Median2 Ha: Median1 > Median2
Ties Included

End:Required+Impacted+No+Slow - Middle:Required+Impacted+No+Slow :
Test Ho: Median(End:Required+Impacted+No+Slow) = Median(Middle:Required+Impacted+No+Slow) vs Ha: Median(End:Required+Impacted+No+Slow) > Median(Middle:Required+Impacted+No+Slow)

	Rank Totals	Cases	Mean Rank
End:Required+Impacted+No+Slow	392	20	19.60
Middle:Required+Impacted+No+Slow	169	13	13
Total	1001	33	30.33
Ties Between Groups	440	28	15.71

U-Statistic: 182
U-prime: 78
Sets of ties between all included observations: 6
Variance: 736.67
Adjustment To Variance For Ties: -47.150
Expected Value: 130
z-Statistic: 1.9803
p = 0.0238
Reject Ho at Alpha = 0.05

Appendix B6 The outstanding work evidence

B6.1 Introduction

This appendix provides detailed evidence to support the analysis presented in chapter eight. The following information is included:

- The classification of each item of outstanding work evidence.
- The results of the Mann Whitney *U* tests of the prevalence for outstanding work during the end of the project rather than during the middle of the project, for the two projects.

B6.2 The classifications of items of outstanding work evidence

Tables B6.2.1 and B6.2.2 present the 'raw' outstanding work evidence for Projects B and C respectively. Each item is labelled with the week in which it occurred in the status meetings. Due to the sensitivity of the information, the entire text for each reference cannot be presented.

Table B6.2.1 Outstanding work evidence for Project B

Week	Phrase	Process Area	Types of outstanding work
6	backlog	Other project	Defects/Fixes
12	outstanding	Test	Performance
13	backlog	Defect Screen Team	Defects/Fixes
14	backlog	Defect Screen Team	Defects/Fixes
14	backlog	Defect Screen Team	Defects/Fixes
15	backlog	Defect Screen Team	Defects/Fixes
16	backlog"	Defect Screen Team	Defects/Fixes
19	outstanding	Development	Design changes
19	outstanding	Test	Defects/Fixes
19	outstanding	Test	Defects/Fixes
19	outstanding	Defect Screen Team	Defects/Fixes
21	outstanding	Test	Problem
22	outstanding	Test	Defects/Fixes
22	outstanding	Other project	Defects/Fixes
22	backlog	Defect Screen Team	Defects/Fixes
24	backlog	Defect Screen Team	Defects/Fixes
26	outstanding	Development	Resource
27	backlog	Other project	Defects/Fixes
29	outstanding	Test	Defects/Fixes
29	backlog	Other project	Defects/Fixes
33	outstanding	Unknown	Action
34	outstanding	Test	Tests
35	outstanding	Test	Tests

Table B6.2.1 Outstanding work evidence for Project B

Week	Phrase	Process Area	Types of outstanding work
36	outstanding	Test	Problem
38	outstanding	Test	Unknown
38	outstanding	Service, NLS	Decision
39	outstanding	Test	Defects/Fixes
39	outstanding	Service, NLS	Decision
39	backlog	Development	Defects/Fixes
40	outstanding	Development	Defects/Fixes
40	outstanding	Information development	Publication items
40	outstanding	Service, NLS	Decision
41	outstanding	Test	Defects/Fixes
41	outstanding	Build	Defects/Fixes
41	outstanding	Information development	Publication items
41	outstanding	Service, NLS	Decision
41	outstanding	Test	Defects/Fixes
41	outstanding	Development	Defects/Fixes
41	backlog	Development	Defects/Fixes
42	outstanding	Service, NLS	Decision
42	outstanding	Other project	Defects/Fixes
42	backlog	Other project	Defects/Fixes
43	outstanding	Development	Defects/Fixes
43	backlog	Other project	Defects/Fixes
44	outstanding	Test	Problem
45	outstanding	Development	Problem
45	outstanding	Information development	Decision
45	outstanding	Test	Tests
45	backlog	Other project	Defects/Fixes
47	outstanding	Information development	Decision
47	outstanding	Test	Problem
47	backlog	Development	Defects/Fixes
48	outstanding	Information development	Decision
48	backlog	Development	Defects/Fixes
48	backlog	Development	Defects/Fixes
49	outstanding	Information development	Decision
49	backlog	Development	Defects/Fixes
50	outstanding	Test	Problem
50	outstanding	Test	Problem
50	backlog	Development	Defects/Fixes
51	outstanding	Early Marketing Support	Problem
51	outstanding	Test	Tests
51	backlog	Development	Defects/Fixes
52	outstanding	Test	Tests
54	outstanding	Test	Tests
54	outstanding	Development	Defects/Fixes
56	outstanding	Test	Tests
56	outstanding	Other project	Defects/Fixes

Table B6.2.2 Outstanding work evidence for Project C

Week	Phrase	Process areas	Type of outstanding work
8	outstanding	Development	Development work
21	outstanding	Development	Development work
26	outstanding	Development	Development work
27	outstanding	Development	Testing
27	outstanding	Development	Secondary changes
27	outstanding	Development	Development work
29	outstanding	External to development team	Development work
30	backlog	Unknown	Service PMRs
32	outstanding	Development	Development work
32	outstanding	Development	Development work
32	outstanding	Development	Problems
33	outstanding	Development	Problems
33	outstanding	Development	Testing
33	outstanding	Development	Development work
34	outstanding	Development	Testing
35	outstanding	Development	Publications
35	outstanding	Test	Defects
36	outstanding	Test	Problems
36	outstanding	Development	Development work
38	outstanding	Development	Documentation
43	outstanding	Test	Defects
43	outstanding	Development	Unknown
45	outstanding	Development	Development work
45	outstanding	Test	Testing
45	outstanding	Test	Testing
45	outstanding	Test	Testing
45	outstanding	Development	Publications
45	outstanding	Unknown	Unknown
45	outstanding	Development	Defects
45	backlog	Test	Problems
46	outstanding	Development	Publications
46	outstanding	Development	Publications
46	backlog	Development	Development work
47	outstanding	Development	Development work
47	outstanding	Information development	Publications
47	outstanding	Development	Documentation
48	outstanding	Development	Development work
48	outstanding	Unknown	Publications
49	outstanding	Development	Development work
49	backlog	Test	Fixes
50	outstanding	Development	Defects
50	outstanding	Development	Problems
50	outstanding	Unknown	Problems
51	outstanding	Development	Defects
51	outstanding	Test	Testing
51	outstanding	Development	Problems
51	outstanding	Unknown	Language translation
51	backlog	Unknown	Defects
51	backlog	Unknown	Defects
51	backlog	Development	Problems

Table B6.2.2 Outstanding work evidence for Project C

Week	Phrase	Process areas	Type of outstanding work
52	outstanding	Development	Development work
52	outstanding	Test	Testing
52	outstanding	Development	Defects
53	outstanding	Development	Defects
53	outstanding	Test	Testing
57	outstanding	Development	Defects
57	outstanding	Development	Development work

B6.3 The Mann Whitney U tests of the prevalence of outstanding work

Presented below are the results of the Mann Whitney *U* tests of the prevalence for outstanding work during the end of the project rather than during the middle of the project, for Projects B and C. The results are taken directly from *Data Desk* v6.0, a data analysis software package for the Apple Macintosh.

Mann Whitney *U* test for Project B

Individual Alpha Level 0.0500
Ho: Median1 = Median2 Ha: Median1 > Median2
Ties Included

End:Number of references - Middle:Number of references :
Test Ho: Median(End:Number of references) = Median(Middle:Number of references) vs Ha: Median(End:Number of references) > Median(Middle:Number of references)

	Rank Totals	Cases	Mean Rank
End:Number of references	769	29	26.52
Middle:Number of references	312	17	18.35
Total	2116	46	46
Ties Between Groups	1035	45	23

U-Statistic: 334
U-prime: 159
Sets of ties between all included observations: 5
Variance: 1930.9
Adjustment To Variance For Ties: -126.82
Expected Value: 246.50
z-Statistic: 2.0601
p = 0.0197
Reject Ho at Alpha = 0.0500

Mann Whitney *U* test for Project C

Individual Alpha Level 0.001
Ho: Median1 = Median2 Ha: Median1 > Median2
Ties Included

End:Number of references per week - Middle:Number of references per week :
Test Ho: Median(End:Number of references per week) = Median(Middle:Number of references per week) vs Ha: Median(End:Number of references per week) > Median(Middle:Number of references per week)

	Rank Totals	Cases	Mean Rank
End:Number of references per week	427.50000	20	21.38
Middle:Number of references per week	133.50000	13	10.27
Total	934	33	28.30
Ties Between Groups	373	25	14.92

U-Statistic: 218
U-prime: 42.5
Sets of ties between all included observations: 4
Variance: 736.67
Adjustment To Variance For Ties: -46.042
Expected Value: 130
z-Statistic: 3.3296
p = 0.0004
Reject Ho at Alpha = 0.001

Appendix B7 Evidence from the feedback workshops

B7.1 Introduction

This appendix complements chapter three, by providing further information on the design and conduct of the feedback workshops.

B7.2 Methodology

As explained in chapter three, the feedback workshops were conducted approximately one year after the completion of the two projects. For Project B, one workshop was conducted (this lasted two hours). For Project C, two workshops were conducted, the second workshop addressing outstanding issues from the first (each of these workshops lasted two hours). For both projects, the respective Project Leader and Project Assistant were present at the workshops.

There were three objectives to the workshops:

- To provide the Project Leaders and Project Assistants with independent assessments of their projects.
- To validate the findings with the Project Leaders and Project Assistants. This primarily consisted of confirming whether the Project Leader and Project Assistant agreed with the findings, and if they did not agree, why they did not agree. Often, Project Leaders and Project Assistants simply wished to provide further information.
- To collect additional information and clarify certain outstanding issues in the research.

Prior to each workshop, the researcher compiled a report of the insights gained for the project, and developed a questionnaire to be filled in by the Project Leader and Project Assistant. The report and the questionnaire complemented each other. The questionnaire provided a method for the Project Leader and Project Assistant to record their opinions of the findings presented in the report. Together, the report and questionnaire provided a structured mechanism for the workshops. In addition to the questionnaire, the three workshops were recorded. This provided a further mechanism for recording information that could not be easily recorded using the questionnaire. It also ensured that the researcher could focus on managing the workshop (e.g. guiding the discussion) rather than taking notes.

The original intention was for the Project Leader and Project Assistant to separately fill in the questionnaires, but the Project Leader and Project Assistant preferred to discuss their answers and supply only one 'aggregated' answer to each of the questions. This was actually an effective approach, because it encouraged the Project Leader and Project Assistant to discuss issues and this provided further useful information for the researcher (which, of course, was recorded on tape).

The reports presented findings in three ways:

- As earlier versions of some of the figures presented in chapter five.
- As earlier versions of some of the figures presented in chapters six through eight.
- As earlier versions of some of the insights summarised in chapter nine.

Overall , the workshops proved to be very effective, satisfying all three of the objectives. The Project Leaders and Project Assistants agreed with most of the findings of the research, wishing to provide additional information rather than correct 'faults' in the findings.

References

[1] Abdel-Hamid, T.K., 'Understanding the "90% Syndrome" in software project management: A simulation-based case study', *Journal of Systems and Software*, 8, pp. 319-330, 1988.

[2] Abdel-Hamid, T.K. and Madnick, S.E., 'Lessons learned from modeling the dynamics of software development', *Communications of the ACM*, 32(12), pp. 1426-1438, 1989.

[3] Abdel-Hamid, T.K., Sengupta, K., and Ronan, D., 'Software project control: an experimental investigation of judgement with fallible information', *IEEE Transactions on Software Engineering*, 19(6), pp. 603-612, 1993.

[4] Abran, A. and Robillard, P.N., 'Function points: A study of their measurement process and scale transformations', *Journal of Systems and Software*, 25(3), pp. 171-184, 1994.

[5] Ballman, K. and Votta, L.G., 'Organizational congestion in large-scale software development', in *Proc. Third International Conference on Software Process*. Reston, Virginia, USA, October: IEEE Computer Society Press, 1994.

[6] Benbasat, I., *An analysis of research methodologies,* in *The Information Systems Research Challenge,* McFarlan, F.W., Editor, Harvard Business School Press: Boston, Massachusetts, 1984.

[7] Benbasat, I., Goldstein, D.K., and Mead, M., 'The case research strategy in studies of information systems', *MIS Quarterly*, 11(3), pp. 369-386, 1987.

[8] Beynon-Davies, P., 'Information systems 'failure': The case of the London Ambulance Service's Computer Aided Despatch project', *European Journal of Information Systems*, 4(3), pp. 171-184, 1995.

[9] Blackburn, J.D., Hoedemaker, G., and Van Wassenhove, L.N., 'Concurrent engineering: prospects and pitfalls', *IEEE Transactions on Engineering Management*, 43(2), pp. 179-188, 1996.

[10] Blackburn, J.D., Scudder, G.D., and Van Wassenhove, L.N., 'Improving speed and productivity of software development: A global survey of software developers', *IEEE Transactions on Software Engineering*, 22(12), pp. 875-885, 1996.

[11] Block, R., *The Politics of Projects*. Yourdon Press: New York, 1983.

[12] Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R., and Selby, R., 'Cost models for future software life cycle processes: COCOMO 2.0', *Annals of Software Engineering*, 1, pp. 57-94, 1995.

[13] Boehm, B.W., *Software Engineering Economics*. Prentice-Hall: Englewood Cliffs, N.J., 1981.

[14] Boehm, B.W., 'Software engineering economics', *IEEE Transactions on Software Engineering*, SE-10(1), pp. 4-21, 1984.

[15] Bonoma, T.V., 'Case research in marketing: Opportunities, problems, and process', *Journal of Marketing Research*, 22(2), pp. 199-208, 1985.

[16] Botting, R.J., 'On the economics of mass-marketed software', in *Proc. 19th International Conference on Software Engineering (ICSE19)*. Boston, Massachusetts, May 17-23: 1997.

[17] Bradac, M.G., Perry, D.E., and Votta, L.G., 'Prototyping a process monitoring experiment', in *Proc. 15th International Conference on Software Engineering*. IEEE Computer Society Press, 1993.

[18] Bradac, M.G., Perry, D.E., and Votta, L.G., 'Prototyping a process monitoring experiment', *IEEE Transactions on Software Engineering*, 20(10), pp. 774-784, 1994.

[19] Brooks, J., F.P., *The Mythical Man-Month*. Anniversary Edition. Addison-Wesley Publishing Company: Reading, Massachusetts, 1995.

[20] Carmel, E., 'Cycle time in packaged software firms', *Journal of Product Innovation Management*, 12(2), pp. 110-123, 1995.

[21] Carmel, E., 'Time-to-completion factors in packaged software development', *Information and Software Technology*, 37(9), pp. 515-520, 1995.

[22] Carr, D. and Koestler, R., 'System dynamics models of software developments', in *Proc. 5th International Software Process Workshop*. Kennebunkport, Maine, USA: IEEE Computer Society Press, 1990.

[23] Cavaye, A.L.M., 'Advice on case studies, for PhD Research (Personal communication)', 6th November 1996.

[24] Cavaye, A.L.M., 'Case study research: A multi-faceted research approach to IS', *Information Systems Journal*, 6(3), pp. 227-242, 1996.

[25] Cook, J.E., Votta, L.G., and Wolf, A.L., 'Cost-effective analysis of in-place software processes', *IEEE Transactions on Software Engineering*, 24(8), pp. 650-663, 1997.

[26] Cook, J.E. and Wolf, A.L., Discovering models of software processes from event-based data. Technical Report No. CU-CS-819-96, Software Engineering Research Laboratory, Department of Computer Science, University of Colorado, 1996.

[27] Cooper, K.G., 'The $2,000 hour: how managers influence project performance through the rework cycle', *Project Management Journal*, 25(1), pp. 11-24, 1994.

[28] Cooper, K.G., 'System dynamics methods in complex project management', in *Proc. NATO Advanced Research Workshop on Managing and Modelling Complex Projects*. Kiev, Ukraine, November 13th-15th: Kluwer Academic Publishers, 1996.

[29] Cuelenaere, A.M.E., van Genuchten, M.J.I.M., and Heemstra, F.J., *Calibrating a software cost-estimation model: Why and how*, in *The Economics of Information*

Systems and Software, Veryard, R., Editor, Butterworth-Heinemann: Oxford, UK, 1991.

[30] Curtis, B., Hefley, W.E., and Miller, S., People capability maturity model (P-CMM). Technical Report No. CMU/SEI-95-MM-02, Carnegie Mellon University/Software Engineering Institute, 1995.

[31] Curtis, B., Krasner, H., and Iscoe, N., 'A field study of the software design process for large systems', *Communications of the ACM*, 31(11), pp. 1268-1287, 1988.

[32] Curtis, B., Walz, D., and Elam, J., 'Studying the process of software design teams', in *Proc. 5th International Software Process Workshop*. Kennebunkport, Maine, USA: IEEE Computer Society Press, 1990.

[33] Dandekar, A., Perry, D.E., and Votta, L.G., 'A study in process simplification', in *Proc. 4th International Conference on Software Process*. Brighton UK: 1996.

[34] Dandekar, A., Perry, D.E., and Votta, L.G., 'A study in process simplification', *Software Process: Improvement and Practice*, 3(2), pp. 87-104, 1997.

[35] Deephouse, C., Mukhopadhyay, T., Goldenson, D.R., and Kellner, M.I., 'Software processes and project performance', *Journal of Management Information Systems*, 12(3), pp. 187-205, 1996.

[36] Deutsch, M.S., 'An exploratory analysis relating the software project management process to project success', *IEEE Transactions on Engineering Management*, 38(4), pp. 365-375, 1991.

[37] Dutoit, A.H. and Bruegge, B., 'Communication metrics for software development', *IEEE Transactions on Software Engineering*, 24(8), pp. 615-628, 1998.

[38] Dutta, S., Kulandaiswamy, S., and Van Wassenhove, L.N., Benchmarking European software management best practices. Working Paper No. 96/45/TM, INSEAD, Fontainebleau, France, 1996.

[39] Eisenhardt, K.M., 'Building theories from case study research', *Academy of Management Review*, 14(4), pp. 532-550, 1989.

[40] Fenton, N., Pfleeger, S.L., and Glass, R.L., 'Science and substance: A challenge to software engineers', *IEEE Software*, 11(4), pp. 86-95, 1994.

[41] Ford, D.N. and Sterman, J.D., 'Dynamic modeling of product development processes', *System Dynamics Review*, 14(1), pp. 31-68, 1998.

[42] Gable, G.G., 'Integrating case study and survey research methods: An example in information systems', *European Journal of Information Systems*, 3(2), pp. 112-126, 1994.

[43] Gasson, S., 'Process modelling of design. (Personal communication)', 25th November 1996.

[44] Gasson, S., 'Process modelling of design. (Personal communication)', 22nd November 1996.

[45] Glaser, B.G. and Strauss, A.L., *The Discovery of Grounded Theory: Strategies for Qualitative Research.* Weidenfield and Nicholson: London, 1967.

[46] Glass, R.L., Vessey, I., and Conger, S.A., 'Software tasks: Intellectual or clerical?', *Information and Management*, 23(4), pp. 183-191, 1992.

[47] Goodhew, P., Ensuring profitable investment in software process improvement - IBM survey results. European Software Process Improvement Foundation, 1996.

[48] Guindon, R., 'Designing the design process: Exploiting opportunistic thought', *Human-Computer Interaction*, 5(2-3), pp. 305-344, 1990.

[49] Guindon, R., 'Knowledge exploited by experts during software system design', *International Journal of Man-Machine Studies*, 33(3), pp. 279-304, 1990.

[50] Heemstra, F.J., 'Software cost estimation', *Information and Software Technology*, 34(10), pp. 627-639, 1992.

[51] Humphrey, W.S., Snyder, T.R., and Willis, R.R., 'Software process improvement at Hughes Aircraft', *IEEE Software*, 8(4), pp. 11-23, 1991.

[52] Jackson, M., 'Problems, methods, and specialisation', *Software Engineering Journal*, 9(6), pp. 249-255, 1994.

[53] Jackson, M., 'Problems, methods, and specialization', *IEEE Software*, 11(6), pp. 57-62, 1994.

[54] Jackson, M., *Software requirements & specification.* Addison-Wesley: 1995.

[55] Jarvenpaa, S.L., 'The importance of laboratory experimentation in IS research', *Communications of the ACM*, 31(12), pp. 1502-1505, 1988.

[56] Jeffery, D.R. and Low, G., 'Calibrating estimation tools for software development', *Software Engineering Journal*, 5(4), pp. 215-221, 1990.

[57] Jenkins, A.M., Naumann, J.D., and Wetherbe, J.C., 'Empirical investigation of systems development practices and results', *Information & Management*, 7, pp. 73-82, 1984.

[58] Kaplan, R.S., The role of empirical research in management accounting. Working Paper No. 9-785-001, Division of Research, Harvard Business School, Boston, Massachusetts, 1985.

[59] Kelly, J.R. and McGrath, J.E., *On Time and Method.* SAGE publications: Newbury, CA, 1988.

[60] Kemerer, C.F., 'An empirical validation of software cost estimation models', *Communications of the ACM*, 30(5), pp. 416-429, 1987.

[61] Kitchenham, B.A., 'Empirical studies of assumptions that underlie software cost-estimation models', *Information and Software Technology*, 34(4), pp. 211-218, 1992.

[62] Kluckhohn, C. and Murray, H.A., ed. *Personality in Nature, Society, and Culture.* Alfred A Knopf: New York, 1948.

[63] Kornreich, T.R. and Smith Parker, S., 'The impact of requirements changes on a large automated information system project: A case study', *Large Scale Systems*, 12(3), pp. 249-256, 1987.

[64] Lapin, L.L., *Statistics for Modern Business Decisions*. Sixth edition. The Dryden Press: Orlando, 1993.

[65] Lederer, A.L. and Prasad, J., 'Information systems software cost estimating: A current assessment', *Journal of Information Technology*, 8(1), pp. 22-33, 1993.

[66] Lee, A.S., 'A scientific methodology for MIS case studies', *MIS Quarterly*, 13(1), pp. 33-50, 1989.

[67] Lee, A.S., 'Advice on case studies, for PhD Research (Personal communication)', 5th November 1996.

[68] Lehman, M.M., Process improvement - the way forward (Revised version of 1995 paper). Report No. MML565, Department of Computing, Imperial College, 1997.

[69] Lehman, M.M., Perry, D.E., and Ramil, J.F., 'Implications of evolution metrics on software maintenance', in *Proc. International Conference on Software Maintenance*. Bethesda, Maryland 16th-20th November: IEEE Computer Society, 1998.

[70] Lehman, M.M., Perry, D.E., and Turski, W.M., Metrics and laws of software evolution - the nineties view. Report No. MML568, Department of Computing, Imperial College of Science, Technology and Medicine, 1997.

[71] Lethbridge, T. and Singer, J., 'Understanding software maintenance tools: some empirical research', in *Proc. Workshop of empirical studies of software maintenance*. Bari, Italy, October: 1997.

[72] Levin, R.I. and Rubin, D.S., *Statistics for Management*. Seventh edition. Prentice-Hall International, Inc.: London, 1998.

[73] MacDonell, S.G., 'Comparative review of functional complexity assessment methods for effort estimation', *Software Engineering Journal*, 9(3), pp. 107-116, 1994.

[74] Maxwell, K., Wassenhove, L.v., and Dutta, S., 'Software development productivity of European space, military and industrial applications', *IEEE Transactions on Software Engineering*, 22(10), pp. 706-718, 1996.

[75] McDermid, J., *Software Engineer's Reference Book*. Butterworth-Heineman: 1991.

[76] McKeen, J.D., 'Successful development strategies for business application systems', *MIS Quarterly*, 7(3), pp. 47-65, 1983.

[77] Merton, R.K., *Social theory and social structure*. Enlarged Edition. The Free Press: New York, 1968.

[78] Miles, M.B., 'Qualitative data as an attractive nuisance: The problem of analysis', *Administrative Science Quarterly*, 24, pp. 590-601, 1979.

[79] Miles, M.B. and Huberman, A.M., *Qualitative Data Analysis*. 2nd. SAGE Publications: Thousand Oaks, CA, 1994.

[80] Mouakket, S., Sillince, J.A.A., and Fretwell-Downing, F.A., 'Information requirements determination in the software industry: A case study', *European Journal of Information Systems*, 3(2), pp. 101-111, 1994.

[81] Myers, M., 'Advice on case studies, for PhD Research (Personal communication)', 6th November 1996.

[82] Nardi, B.A. and Engestrom, Y., 'A web on the wind: the structure of invisible work', *Computer Supported Cooperative Work*, 8(1-2), pp. 1-174, 1999.

[83] Olsen, N.C., 'The software rush hour', *IEEE Software*, 10(5), pp. 29-37, 1993.

[84] Olsen, N.C., 'Survival of the fastest: Improving service velocity', *IEEE Software*, 12(5), pp. 28-38, 1995.

[85] Olson, G.M., Olson, J.S., Carter, M.R., and Storrosten, M., 'Small group design meetings: An analysis of collaboration', *Human-Computer Interaction*, 7(4), pp. 347-374, 1992.

[86] Orlikowski, W., 'Seeking advice on design of case studies (Personal communication)', 8th November 1996.

[87] Orlikowski, W.J., 'CASE tools as organizational change: Investigating incremental and radical changes in systems development', *MIS Quarterly*, 17(3), pp. 309-340, 1993.

[88] Paulk, M., Curtis, B., Chrissis, M., and Weber, C., Capability maturity model for software (version 1.1). Technical Report No. CMU/SEI-93-TR-024, Carnegie Mellon University/Software Engineering Institute, 1993.

[89] Paulk, M., Weber, C., Garcia, S., Chrissis, M.B., and Bush, M., Key practices for the capability maturity model (version 1.1). Technical Report No. CMU/SEI-93-TR-025, Carnegie Mellon University/Software Engineering Institute, 1993.

[90] Paulk, M.C., Curtis, B., Chrissis, M.B., and Weber, C.V., 'Capability maturity model, version 1.1', *IEEE Software*, 10(4), pp. 18-27, 1993.

[91] Perry, D.E., Staudenmayer, N.A., and Votta Jr., J.G., *Understanding and improving time usage in software development,* in *Trends in software: software process,* Fuggetta, A. and Wolf, A.L., Editor, John Wiley and Sons Ltd: 1995.

[92] Perry, D.E., Staudenmayer, N.A., and Votta, L.G., 'People, organizations, and process improvement', *IEEE Software*, 11(4), pp. 36-45, 1994.

[93] Perry, D.E. and Votta, L.G., The planning number 2.5 +,-.5. Technical Report No. BL0112650-930930-28TM, AT&T Bell Laboratories, Murray, NJ, 1993.

[94] Pettigrew, A.M., 'Longitudinal field research on change: Theory and practice', *Organization Science*, 1(3), pp. 267-292, 1990.

[95] Phalp, K., An evaluation of software modelling in practice. Doctoral thesis, Bournemouth University, 1995.

[96] Phan, D.D., Vogel, D.R., and Nunamaker Jr., J.F., 'Empirical studies in software development projects: Field survey and OS/400 study', *Information and Management*, 28(4), pp. 271-280, 1995.

[97] Putnam, L.H., 'A general empirical solution to the macro software sizing and estimating problem', *IEEE Transactions of Software Engineering*, SE-4(4), pp. 345-361, 1978.

[98] Quintas, P., 'Engineering solutions to software problems: some institutional and social factors shaping change', *Technology Analysis and Strategic Management*, 3(4), pp. 359-376, 1991.

[99] Rainer, A. and Shepperd, M.J., A framework for investigating software project schedule behaviour. Technical Report No. ESERG: TR98-003, Bournemouth University, 1998.

[100] Rainer, A.W., M. J., An empirical investigation into waiting in software development projects. Technical Report No. ESERG TR98-008, Bournemouth University, 1998.

[101] Remenyi, D. and Williams, B., 'Some aspects of methodology for research in information systems', *Journal of Information Technology*, 10(3), pp. 191-201, 1995.

[102] Rodden, T., King, V., Hughes, J., and Sommerville, I., 'Process modelling and development practice', in *Proc. Third European Workshop on Software Process Technology (EWSPT'94)*. Villard de Lans, France: Springer-Verlag, 1994.

[103] Rodrigues, A. and Bowers, J., 'The role of system dynamics in project management', *International Journal of Project Management*, 14(4), pp. 213-220, 1996.

[104] Rodrigues, A. and Bowers, J., 'System dynamics in project management: a comparative analysis with traditional methods', *System Dynamics Review*, 12(2), pp. 121-139, 1996.

[105] Rodrigues, A.G. and Williams, T.M., System dynamics in software project management: towards the development of a formal integrated framework. Theory, Method and Practical Science No. 96/5, Department of Management Science, University of Strathclyde, 1996.

[106] Schofield, C., An empirical investigation into software estimation by analogy. Doctoral thesis, Bournemouth University, 1998.

[107] Schriber, J.B. and Gutek, B.A., 'Some time dimensions of work: measurement of an underlying aspect of organization culture', *Journal of Applied Psychology*, 72(4), pp. 642-650, 1987.

[108] Sengupta, K. and Abdel-Hamid, T.K., 'The impact of unreliable information on the management of software projects: A dynamic decision perspective.', *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 26(2), pp. 177-189, 1996.

[109] Shaw, M., 'Prospects for an engineering discipline of software', *IEEE Software*, 7(6), pp. 15-24, 1990.

[110] Shepperd, M., 'Products, processes and metrics', *Information and software technology*, 34(10), pp. 674-680, 1992.

[111] Shepperd, M., Schofield, C., and Kitchenham, B., 'Effort estimation using analogy', in *Proc. 18th International Conference on Software Engineering (ICSE-18)*. Berlin, Germany, March 25-29: 1996.

[112] Singer, J., 'Practices of software maintenance', in *Proc. International Conference on Software Maintenance*. Bethesda, Maryland 16th-20th November: IEEE Computer Society, 1998.

[113] Singer, J., Lethbridge, T., Vinson, N., and Anquetil, N., 'An examination of software engineering work practices', in *Proc. Centre for Advanced Studies Conference (CASCON'97)*. Toronto, November: 1997.

[114] Smith, P.G. and Reinertsen, D.G., *Developing products in half the time*. Updated paperback edition. International Thomson Publishing Inc.: New York, 1995.

[115] Soloway, E. and Iyengar, S., *Empirical studies of programmers*. Schneiderman, B. (Ed.) Ablex Publishing Corporation: Norwood, New Jersey, 1986.

[116] Sommerville, I. and Monk, S., 'Supporting informality in the software process', in *Proc. Third European Workshop on Software Process Technology (EWSPT'94)*. Villard de Lans, France: Springer-Verlag, 1994.

[117] Sommerville, I. and Rodden, T., 'Understanding the software process as a social process', in *Proc. Second European Workshop on Software Process Technology (EWSPT'92)*. Trondheim, Norway: Springer-Verlag, 1992.

[118] Sommerville, I. and Rodden, T., Human, social and organisational influences on the software process. Technical Report No. CSEG/2/1995, Lancaster University, 1995.

[119] Stone, E., *Research Methods in Organizational Behaviour*. Scott, Foreman, and Company: Glenview, Illinois, 1978.

[120] Strauss, A. and Corbin, J., *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*. SAGE Publications: Newbury Park, California, 1990.

[121] Taff, L.M., Borchering, J.W., and Hudgins Jr., W.R., 'Estimeetings: Development estimates and a front-end process for a large project', *IEEE Transactions on Software Engineering*, 17(8), pp. 839-849, 1991.

[122] Tate, G. and Verner, J., *Software Costing in Practice,* in *The Economics of Information Systems and Software,* Veryard, R., Editor, Butterworth-Heinemann: Oxford, UK, 1991.

[123] Thamhain, H.J. and Wilemon, D.L., 'Criteria for controlling projects according to plan', *Project Management Journal*, 17(2), pp. 75-81, 1986.

[124] Tufte, *Envisioning Information.* Graphics Press: 1991.

[125] Tufte, E.R., *The Visual Display of Quantitative Information.* Graphics Press: 1983.

[126] Tvedt, J.D., An extensible model for evaluating the impact of process improvements on software development cycle time. Doctoral thesis, Arizona State University, 1996.

[127] Tvedt, J.D. and Collofello, J.S., 'Evaluating the effectiveness of process improvements on software development cycle time via system dynamics modeling', in *Proc. International Computer Software and Applications Conference (CompSAC'95).* 1995.

[128] van Genuchten, M., 'Why is software late? An empirical study of reasons for delay in software development', *IEEE Transactions on Software Engineering*, 17(6), pp. 582-590, 1991.

[129] Verner, J. and Tate, G., 'A software size model', *IEEE Transactions on Software Engineering*, 18(4), pp. 265-278, 1992.

[130] Waeselynck, H. and Pfahl, D., 'System dynamics applied to the modelling of software projects', *Software - Concept and Tools*, 15(4), pp. 162-176, 1994.

[131] Walsham, G., 'Interpretive case studies in IS research: Nature and method', *European Journal of Information Systems*, 4(2), pp. 74-81, 1995.

[132] Walz, D.B., Elam, J.J., and Curtis, B., 'Inside a software design team: Knowledge acquisition, sharing, and integration', *Communications of the ACM*, 36(10), pp. 63-77, 1993.

[133] Waterson, P.E., Clegg, C.W., and Axtell, C.M., 'The dynamics of work organisation, knowledge, and technology during software development', *International Journal of Human-Computer Studies*, 46(1), pp. 79-101, 1997.

[134] Watson, K.I., 'COCOMO as a schedule prognosis and validation tool: a case study', *Software Quality Journal*, 1(4), pp. 193-208, 1992.

[135] Wohlin, C. and Ahlgren, M., 'Soft factors and their impact on time to market', *Software Quality Journal*, 4(3), pp. 189-205, 1995.

[136] Wohlwend, H. and Rosenbaum, S., 'Schlumberger's software improvement program', *IEEE Transactions on Software Engineering*, 20(11), pp. 833-839, 1994.

[137] Wolcott, H.F., *The Art of Fieldwork*. Altimira Press: Walnut Creek, California, 1995.

[138] Wolf, A.L. and Rosenblum, D.S., 'A Study in Software Process Data Capture and Analysis.', in *Proc. Second International Conference on The Software Process*. Berlin, Germany.: IEEE Computer Society Press, Los Alamitos, California, 1993.

[139] Yin, R.K., *Case Study Research, Design and Methods*. 1st edition. SAGE Publications: Beverly Hills, CA, 1984.

[140] Yin, R.K., *Case Study Research: Design and Methods*. 2nd edition. SAGE Publications: 1994.

[141] Ziman, J., *Reliable Knowledge: An Exploration of the Grounds for Belief in Science*. Cambridge University Press: 1978/1991.