Re-planning for a Successful Project Schedule

Austen Rainer and Martin Shepperd
Empirical Software Engineering Research Group
Bournemouth University
Talbot Campus
Poole, BH12 5BB, UK
{arainer, mshepper}@bournemouth.ac.uk

Abstract

Time to market or project duration has increasing significance for commercial software development. We report on a longitudinal study of a project at IBM Hursley Park. The focus of this study was schedule behaviour, however, we explored a range of related factors including planned versus actual progress, resource allocation and functionality delivered. In the course of the twelve-month study, evidence was collected from 8 interviews, 49 project meetings, a number of other project documents and a feedback workshop. The project leader considered the project to be a success, not only in terms of satisfying resource and schedule objectives, but also in the marketplace. Whilst many of the originally planned external commitments were met, it is clear that the project did not adhere to its original (detailed) plan and indeed there were no less than seven re-plans. These re-plans were mainly in response to misestimates in the original plan, rather than in response to the introduction of additional requirements (of which there were several) or problems with external dependencies. Furthermore, these re-plans suggest a distinction between the nature of the initial planning process and the nature of the re-planning process during the project. Attention is also directed at the implications these re-plans have for metrics and estimation research.

Keywords: successful software project, case-study, schedule, metrics, planning, estimation

1. Introduction

A growing body of research reports on the actual behaviour of software development projects. Most of these studies focus on low level processes, such as the time usage of individual developers [4, 14, 15], the progress of activities [9, 21] and the cognitive processes of individual designers [10, 11]. There are few studies that have systematically investigated the high-level processes that occur within software projects, such as the behaviour of process areas of the project, or the interaction between low-level and high-level processes. Notable exceptions are Curtis *et al*. [6-8] and the system dynamics work of Abdel-Hamid and his colleagues [2]. Overall, these studies, whether they investigate the high-level or low-level processes, tend not to evaluate the impact of the studied process(es) on schedule behaviour (although, again, the work of Abdel-Hamid is a notable exception).

It is already established that project managers tend to adjust their project schedules in response to changes in their project. For example, Sommerville writes:

> "Project managers revise the assumptions about the project as more information becomes available. They re-plan the project schedule." [19], p. 50)

Rook [18] 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." [13], chapter 27 page 19)

And, from a different perspective, Rodden *et al*. [17] write:

> "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." [17], p. 61)

Re-plans (or adjustments) are necessary for a number of reasons, such as:

1. Events which were unexpected and unplanned for. One example is the departure of key personnel to a different organisation.
2. Events which were expected but cannot be planned for. Examples here are working overtime and working shifts: managers and developers expect that these will be necessary, but cannot (or do not) explicitly plan for *when* they will occur. In other situations, contingency can still be built into the plan. An example of this strategy is the expectation that there will be changes to the requirements: managers may plan time and effort for those changes, but they might not be able to plan for *when* these changes will occur, or *what* precisely those changes will be.
3. Events which were expected and planned for, but for which the original plans were inaccurate. Examples include inaccurate estimates and an incomplete or inaccurate work breakdown structure.

All of these events may occur through a combination of high-level and low-level processes, which are internal or external to the project. As already noted, there is a lack of evidence on high-level project processes and their interaction with low-level processes. As a result, although the occurrence of re-plans is recognised, detailed empirical studies on the nature and implications of these re-plans do not appear to have been conducted. Consequently, there is a clear need for further investigation, for the study of how a project's management react to changes in the project (caused by these processes), and a consideration of the effects of these processes and their 'associated' re-plans on the schedule behaviour of the project.

We have conducted two case studies based upon real-world software development projects, Project B and C. This paper focuses upon Project B, with some references to Project C. The case studies were unusual in three ways. First, they adopted a holistic approach, considering both high and low-level processes in the projects. As already noted, Curtis *et al*. [7] adopted a similar approach, but focused on different processes to those considered here. Curtis *et al*. investigated human behaviour with a simple hierarchy of behavioural levels, whereas this study has investigated 'project behaviour', using such concepts as workload, capability and schedule.

Second, the case studies considered the dynamic behaviour of processes: how they changed over time. Abdel-Hamid and his colleagues (e.g. [2]) have investigated dynamic behaviour. Whereas Abdel-Hamid focused on validating their systems' dynamics models, this investigation has concentrated on exploring the underlying phenomena.

Third, the case studies combined multiple sources of naturally occurring evidence. Cook *et al*. [5] used naturally occurring evidence in their investigations. Whereas Cook *et al*. primarily rely upon electronically 'mining' quantitative sources of evidence, our case studies employed a semi-autonomous method (i.e. a combination of electronic text searches and manual 'browsing' of the evidence) to identify, extract and organise qualitative and quantitative information.

2. The Case Study

Project B is one release of a mission-critical, middleware transaction processing system (referred to as Product B in this paper) that operates on mainframe computers. Other versions within the 'family' operate on mid-range machines and workstations. Overall, Project B was considered a success by its Project Leader. As one criterion of this success, the release was "delivered" when planned. Closer inspection of the project, however, indicates that two features were not delivered with the product (but were delivered some weeks later, via the World Wide Web), and that the quality of one of these features was much lower than desired by the project team. Also, an important feature was developed externally to Project B (but within the organisation) but tested by Project B. This introduced significant external dependencies. Nevertheless, the feature was delivered with the product without presenting any serious problems to the progress of the project.

Table 1 summarises the sources of evidence that were collected for Project B. As the table indicates, naturally occurring evidence was supplemented by interviews, a feedback workshop following the completion of the project, and a number of other documents. The feedback workshop 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 [21] adopted a similar approach in his study.

Table 1 Evidence collected from Project B

Type of evidence	Count
Interviews	8
Meeting minutes, of which:	51
- Project status meetings	49
- Senior management meetings	1
- Project review (post-mortem)	1
Researchers records of status meetings	2[1]
Project schedules	1
Projector overheads (from presentations)	1
Project documents, of which:	6
- Plans	3
- Other documents	3
Risk assessments	2
Project 'contract'	1
Feedback workshop questionnaires	1
Total number of 'documents'	73

Table 2 Summary of Project B

Characteristic	Project B
Size of development team	approx. 38 people
Size of management team	approx. 6 people
Strategic value of the product	High; long-term
Type of product	Large, mission-critical, middleware, transaction-processing, legacy system
Purpose of project	New functionality
Size of changes in this release	36 KLOC[2] of new code
Project duration (in weeks)	57 (planned and actual)
Product delivery date	Week 52 (planned and actual)
Determination of project duration	Project end-date driven, due to market considerations
Composition of management team	The project used a multi-functional project management team, with representatives from each significant process area.

[1] 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 represented the actual content of those meetings. The 'learning curve' required to understand the discussions at the meetings meant that this approach was unfeasible, and consequently it was not pursued. The inability to independently assess the representiveness of the minutes is recognised as a threat to the validity of this investigation.

[2] Given the size of the product (36KLOC) and the amount of effort, it may appear that Project B was not particularly productive. As already stated, Product B is a mission-critical, middleware processing system, which is also a legacy system. Consequently, there are stringent quality requirements for this product, and also additional problems associated with enhancing a legacy system.

The primary source of evidence used in this analysis was the minutes of project status meetings, and these were supplemented by information from interviews, project schedules and the feedback workshop. Project status meetings were the highest-level meetings within the project and occur regularly (typically weekly or fortnightly). They were attended by representatives from all the process areas (e.g. Design/Code, Test, Marketing, Finance, Service) of the project, and were a naturally occurring phenomenon (so that the researcher did not intrude on the project). Project status meetings normally lasted between 1.5 to 2 hours, each producing about 10 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. Design changes were 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. Interestingly, the minutes did 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 were made but not recorded (note, however, that no such discussion occurred at the two meetings attended by the researcher). Another possibility was that the comparisons occurred outside the project status meeting (which would be surprising since the project status meeting was an explicit mechanism for reporting the progress of each process area to the rest 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 meeting 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 type of information (i.e. dynamic process information) that was unavailable from other sources of evidence.

Table 2 summarises some of the characteristics of the project. Note that the development team includes design/code, test and support personnel (e.g. build and library control systems). The management team includes management and 'support management'.

Figure 1 presents the planned and actual project-level schedule (phases and milestones), the planned and actual feature-level schedule for two features (features F02 and F03), all of the plans and re-plans for the project, and a number of events (mostly referring to these two features) that occurred during the project.

For the project-level schedule and the plans and re-plans, small solid squares represent internal milestones or internal re-plans (i.e. milestones or re-plans that do not require interaction with senior management) and circles represent external milestones or external re-plans (i.e. milestones or re-plans that do require interaction with senior management). The thin horizontal lines represent planned variation in when a milestone and/or a phase may complete (so that, for example, the Design/Code phase is planned to complete between week 19 and week 23). The broken arrow-lines indicate which features the internal re-plans refer to. The acronyms D, FV and ST represent Design/Code, Functional Verification and System Test respectively. A question mark indicates uncertainty as to the commencement or completion of a phase or re-plan.

The Design/Code phase is concerned with the high-level design, low-level design, coding, unit testing, and fixing of defects discovered in the unit testing. The Functional Verification phase is concerned with testing the proper execution of a distinct function, such as an API (Application Programming Interface) call or a GUI (Graphical User Interface) menu. The System Test phase is concerned with testing the complete product within its operating environment, with other products.

Only information on features F02 and F03 are included in the figure because these were the only features that were substantially referenced in the meeting minutes. Presenting information on only two features might appear to bias the analysis of Project B but, in fact, the reason these features were substantially referenced in the minutes is precisely because they are the most problematic features on the project