Representing the behaviour of software projects using multi-dimensional timelines

Austen Rainer
School of Computer Science
University of Hertfordshire
College Lane Campus
Hatfield
Hertfordshire AL10 9AB
U.K.
a.w.rainer@herts.ac.uk

Context: There are few empirical studies in the empirical software engineering research community that describe software projects, at the level of the project, as they progress over time.

Objective: To investigate how to coherently represent a large volume of qualitative and quantitative data on a range of project-level attributes as those attributes change over time.

Method: Develop a modelling technique, multi-dimensional timelines (MDTs) and undertake a preliminary appraisal of the technique using examples from a longitudinal case study of a project at IBM Hursley Park.

Results: MDTs can represent project-level attributes as they change over time, provided these attributes, and the empirical data about them, can be located in time (an analytical requirement) and can be represented in terms of the simple geometrical structures of points, lines and planes (a graphical requirement). Changes in attributes are documented at the point in time at which the change occurs. There are a number of ways in which an attribute can be represented on the MDT: as a quantitative time series, as an event, as an event with label containing complex qualitative information, or as a schedule. The MDT technique is currently not capable of representing relationships between different attributes e.g. a causal relationship.

Conclusion: The initial appraisal of MDTs is encouraging, but further work is needed on the development of the MDT technique and on its evaluation.

Keywords: case study, software project, multi-dimensional timeline, time series, qualitative data, quantitative data

1 Introduction

The widespread occurrence of software projects, together with the almost perennial concern as to the success and failure of such projects (e.g. [1-8]) is evidence of the importance of projects to the IT and software industries. In addition, a software project occurs over a period of time and changes during that period, and intuitively one would expect projects with longer duration to be more susceptible to change. Despite the importance of software projects, and the fact that they change as they progress, there is surprisingly little research in the empirical software engineering research community that has investigated actual software projects at the level of the project and as the project changes over time. (In section 2 we review previous research that has investigated actual projects over time.) This lack of investigation constrains our understanding of software engineering and software engineering management.

Whilst there is a need to study projects at the level of the project, and as the project changes over time, it is extremely challenging to conduct such studies. These studies are time-consuming and effortful and hence costly. They produce large volumes of structured and unstructured data, much of it generated by the project, and large quantities of this data would be need to be pre-processed (e.g. qualitative data would need to be coded [9]) before it could be analysed using conventional approaches (e.g. statistical software). In addition to the large volumes of data there would also be a large, complex and even contradictory set of conceptualisations to consider, both those used by participants in the project and those used by the researchers in their research programmes.

In this paper, we focus on one challenge: the concise representation of large volumes of diverse data collected on a range of project-level attributes as those attributes change over time. We propose multi-dimensional timelines (MDTs) as a technique for tackling that challenge and we use examples from a 58-week exploratory, longitudinal case study of a software development project at IBM Hursley Park to demonstrate the use of the technique. The MDTs draw on a variety of types of evidence to, for example, present quantitative data (e.g. number of design changes), event data (e.g. decisions) and schedule data (e.g. durations and checkpoints).

We organise the remainder of this paper as follows: section 2 reviews related research on the study of software project-level attributes over time, and the use of techniques to represent changes over time. Section 3 introduces and briefly explains the multi-dimensional timeline technique. Section 4 describes the design of the preliminary appraisal of the technique. Section 5 presents and discusses three examples of MDTs. Section 6 then discusses the results of the appraisal and considers directions for further research.

2 Related work

2.1 The investigation of software project behaviour over time

There have been a number of studies over the years that have investigated software projects, but there have been few studies that have investigated the changes in project-level attributes over time of *actual* projects. Starting with Abdel-Hamid and Madnick's seminal work [10] developing a system dynamics model of software projects, there has been a growing interest in approaches to *simulating* the behaviour of software projects over time (e.g. the series of *International Workshops on Software Process Simulation and Modeling*, organised by Portland State University, USA) but work in this area tends not to follow the progress of an actual project over time e.g. with a fine-grained longitudinal study. Another body of research investigates the critical success factors for successful software projects [11], but again work in this area tends not to follow the progress of actual projects over time, but instead relies on survey studies (e.g. [12]) or analyses of previously collected datasets on project attributes (such as van Genuchten's review of project plans [13]). Yet another body of work investigates the parts of actual software projects. For example, in another classic study, Curtis et al. [14] investigated the software design process, considering five behavioural layers of software design: the individual, the team, the project, the organisation and the business milieu. Curtis et al.'s study was based on interviews with participants on 17 projects, again not investigating behaviour over time. Rodden et al. [15] has compared process models with actual development. Waterson et al. [16] have studied the impact of organisational and cognitive factors on the progress of commercial projects. Hesse et al. [17] has also investigated actual behaviour of parts of projects against prescribed behaviour but again not over time.

The lack of study of software project behaviour over time constrains our understanding of software engineering and software engineering management, and does so in at least three ways. First, it is difficult for findings from studies of part of a software project to be properly appreciated within the context of a whole project. This is indirectly recognised by researchers when they recognise the potential impact of context on the results of their studies e.g. [18]. For example, it is generally recognised that software code inspections are a cost effective method of identifying defects in software code, but there are many projects that do not use software code inspections, and there may be certain types of project behaviour that make using software code inspections difficult e.g. effort, resource and time constraints. Second, it is difficult to develop a theory of software projects and their actual behaviour, and to then test that theory. Third, there is a 'vicious circle' in terms of methodology and research effectiveness: until we study project-level attributes as they change over time we cannot learn how *best* to study project-level attributes as they change over time; but it is also hard to judge which project-level attributes are most valuable to study.

The 'vicious methodological circle' is perhaps reflected in the methodological advice on the conduct of empirical studies. A number of recent papers [9, 19-22] advise on how to study software behaviour and technologies. These papers tend to treat time either as a property of the phenomenon being studied (e.g. the lead time of a project) or of the study itself (e.g. the time to undertake the study). But the advice does not extend to consideration of how to study a phenomenon *over* time. As a first example, Runeson and Höst [19] refer to Yin's [23] variety of alternative structures for *reporting* case studies, of which one structure is *chronological*. Besides recognising the chronological structure for reporting the results of a case study, Runeson and Höst direct little attention at how to analyse a phenomenon over time, or how to report findings involving time. As a second example, Seaman [9] acknowledges that "Historical qualitative information can also be gained by examining documentation" ([9]; p. 558); but other than recognising historical information, Seaman's article does not explicitly address the analysis or reporting of behaviour over time.

2.2 The use of timelines in empirical software engineering

Whilst empirical software engineering researchers appear to lack an awareness of the importance of investigating project behaviour over time, they do clearly consider time when investigating phenomenon within a project. Bradac et al.'s work ([24] but see also [25]) is particularly relevant here because it is one of the few studies we are aware of that reports a time line analysis of coded-qualitative data. Bradac et al.'s study analysed a time diary maintained by a developer and the paper presents a simple time line over 75 consecutive days indicating the task being undertaken by the developer (e.g. plan development, design, code) and the state of that task (broadly, whether the developer was able to work on the task, or was waiting on some input for that task). Walz et al. [26] report a simple time line of project staffing together with a small number of "major events" over the four months during which they videotaped the 19 design team meetings for their study.

A separate study also of relevance here is the work undertaken by Guindon [27, 28] to understand the cognitive work undertaken by developers. Guindon used protocol analysis, with two developers verbalising their thinking as they tackled a design problem. Guindon then classified the verbalised reports, presented them as 90-minute time series, and compared the classification of actual cognitive work against the prescribed work. She used the results to argue that developers undertake an opportunistic approach to design rather than a structured, linear approach i.e. moving opportunistically between modelling the problem, understanding the requirements, and high, medium and low level solutions. Others (e.g. [29, 30]) have subsequently challenged Guindon's claims and proposed alternative explanations; the important point here however is the way that Guindon has used a time-series of categories of data, comparing work over time against prescribed phases, to underpin her arguments.

Cook et al.’s work ([31]; see also [32, 33] for related work) is helpful here as a point of contrast. Cook et al. exploit readily available, existing data sources to support the historical analysis of an in-place software process. Cook et al.’s approach differs to ours in a number of respects however. They focus on a process lower than the level of the project and on quantitative data only; and there is therefore no explicit representation that integrates quantitative and qualitative data in a time line. Also, while they collected data over time they do not report that data as a time series, although they do report a finite-state machine model of the process. In section 5 of this paper, we take examples from one instance (a project) whilst Cook et al. report on a large number of instances. Cook et al. recognise that it may not be easy to integrate data from different sources. In the examples reported here, we use the relative week number of the project as an index for integration of different data sources and different types of data.

There is a wide range of other studies that examine actual processes within software projects, but these studies do not seem to report findings in terms of time lines involving qualitative and quantitative data, or even quantitative time-series e.g. [14, 16, 17, 34-44]. As other examples: Heijstek and Chaudron [45] visualise effort distributions for different stages in projects; and Wnuk et al. [46] visualise the inclusion and exclusion of candidate features from the next release of a product. Neither of these studies includes qualitative data, nor combines qualitative and quantitative data.

2.3 The use of timelines in other research disciplines

Other research disciplines recognise the importance of investigating a phenomenon over time. We briefly focus on three areas of research: case study design [23], experimental and quasi-experimental design [47], and qualitative data analysis [48]. Overall, whilst these areas recognise the importance of investigating behaviour over time, these techniques are limited in various ways: they may be one dimensional, or focus only on quantitative data, or not represent time at a uniform level of granularity, or not consider time at a sufficiently low level of abstraction.

2.3.1 Case study design

Yin [23] suggests six structures for reporting case studies. Yin considers the *chronological* structure to be relevant to explanatory, descriptive or exploratory research; recognises that case studies “generally cover events over time” ([23]; p. 153); and also recognises that chronological structures are important because “presumed causal sequences must occur linearly over time” ([23]; p. 153). Yin highlights one pitfall to chronological structures: disproportionate attention is usually given to the early events with insufficient attention directed at the later events.

2.3.2 Experimental and quasi-experimental design

Shadish et al. [47] devote a chapter of their book to discuss the interrupted time series design. For Shadish et al., time series refers to a large series of observations made on the same variable consecutively over time. The observations may be made on the same unit, or on different but similar units. Interrupted time series allows the investigator to study the effect of a treatment that occurred at a known point in time. If the treatment had an effect, then the observations after the treatment should ‘behave’ in a different way to the observations before the treatment. Implied within Shadish et al.’s discussion is the presence of events e.g. the introduction of the treatment. As the title of their book indicates, Shadish et al. examine time series from the perspective of experimental or quasi-experimental designs. In contrast, we are interested in time *lines* that combine qualitative and quantitative data, and do so from the perspective of case study design which means that we have inevitably small sample sizes. These contrasting perspectives may explain our broader and perhaps more accommodating concept of time lines rather than Shadish et al.’s time series.

2.3.3 Qualitative data analysis

Miles and Huberman [48] present a sourcebook of techniques and ideas for qualitative data analysis. This sourcebook includes a range of techniques for "time-ordered displays" to aid within-case and cross-case exploration and description. Miles and Huberman go on to explain that, for the qualitative researcher, narratives are probably indispensable if we are to understand a complex chronology in its full richness; but the problem is that going straight to an extended narrative from written-up field notes runs the risk that the narrative is partial, biased, or simply "dead wrong" ([48]; p. 111). They write:

> "Life is chronology. We live in a flow of *events*... the problems we face in understanding event flows are those of sorting out the different domains of events, preserving the sequence, showing the salience or significance of preceding events on following events – and doing all of this in an *easily visible display* that lets us construct a valid chronology." ([48]; p. 111; emphasis added)

In their sourcebook, Miles and Huberman present and discuss a range of techniques, the main ones being event listing, critical incident chart, time-ordered matrix and time-ordered meta-matrix. For each of these, Miles and Huberman present examples and advice on the application of the technique. We summarise this advice in Table 1.

Table 1 Summary of four techniques for representing qualitative data over time

Type	Comments
Event list (EL)	**Summary**: A matrix that arranges a series of concrete events by chronological time periods, sorting them into several categories. **Benefit**: ELs are easy to do and have good payoff in understanding any extended process. They can be done at many different levels of detail. **Time required**: varies between 15 minutes to half a day to 5 hours of work
Critical incident chart and timeline (CICT)	**Summary**: ELs that are limited more sharply to critical incidents (e.g. incidents that are important or crucial) and/or to an immediate setting **Benefits**: not stated, but the implication is that CICTs are useful for when one wants to focus on critical, influential or decisive events. **Time required**: not stated
Time-ordered matrix (TOM)	**Summary**: A more general form of the ELs. Has its columns arranged by time period, in sequence, so that the investigator can see when particular phenomena occurred. The basic principle is chronology. The rows depend on what else the investigator is studying. **Benefits**: use a descriptive TOM when the data is fairly complete, to begin developing possible explanations that can be tested **Time required**: if field notes have been coded, a TOM could be assembled in 2 – 3 hours, but more complex TOMs will take longer.
Time-ordered meta-matrix (TOMM)	**Summary**: Columns organised sequentially by time period; the rows need not be ordered, but can have the cases in arbitrary order. Again, the basic principle is chronology. **Benefits**: TOMMs are easy to do and work best when one wants to examine a reasonably specific class of time-related events (or states) across cases. **Time required:** generally speaking, if one spends adequate time in specifying the events and developing code, the analysis can go relatively quickly. In one example provided, it took 3 hours to read the case reports, another 3 hours to enter the data in a refined matrix, and then another hour for drawing conclusions and writing up.

.

2.4 Summary of the preceding review

Software projects and their behaviour are very important however the change in projects over time is not being investigated by the empirical software engineering research community. Indeed, on the basis of the methodological advice published, it seems that the community lacks an awareness of the importance of software project behaviour over time. Other fields of inquiry recognise the importance of investigating behaviour over time and have presented a number of techniques for representing this behaviour however these techniques are often limited in some way: they focus on quantitative data rather than qualitative data; may not present multiple variables concurrently in time (instead focusing on, for example, single time series), or do not maintain a consistent, low-level temporal granularity. The investigation of software projects is extremely challenging in terms of (at least) the demand on resources, the scale and complexity of the data that would be collected and analysed, as well as the scale and complexity of possible conceptualisations of the projects. Indeed, these challenges may help to explain the limited number of studies of software project behaviour over time.

One particular challenge is the concise representation of large volumes of diverse data collected on a range of project-level attributes as those attributes change over time. This challenge motivates the work we report here i.e. on the preliminary development of multi-dimensional timelines (MDTs) as a technique for tackling that challenge.

3 Multi-dimensional timelines

A multi-dimensional timeline (MDT) is intended to visualise multiple sets of data over time with in general each set of data being of a particular type e.g. a quantitative time-series, a schedule of phases, a sequence of events, or some qualitative data. As we will show with the examples in section 5, MDTs require data that can be represented in time and that can be represented using the simple geometrical structures of points, lines and planes. The first of these requirements is, in a sense, an analytical constraint; the second of these requirements is a graphical design constraint.

Broadly speaking an MDT is structured into one or more sections in the visual display. Each section provides a graphical space to represent a set of data. A set of data is positioned in that space in a way that is sensible for that type of data. For example, a quantitative time series could be positioned in the space as a histogram, whereas a set of decisions may be positioned as a sequence of events. The MDTs in section 5 provide a range of examples of how types of data are positioned in the sections/spaces. Whilst each section possesses its own y-axis (where the meaning of that y-axis is specific to the type of data), all sections share a common x-axis (time) which is needed in order to represent data as events in time, and to synchronise different information in the MDT.

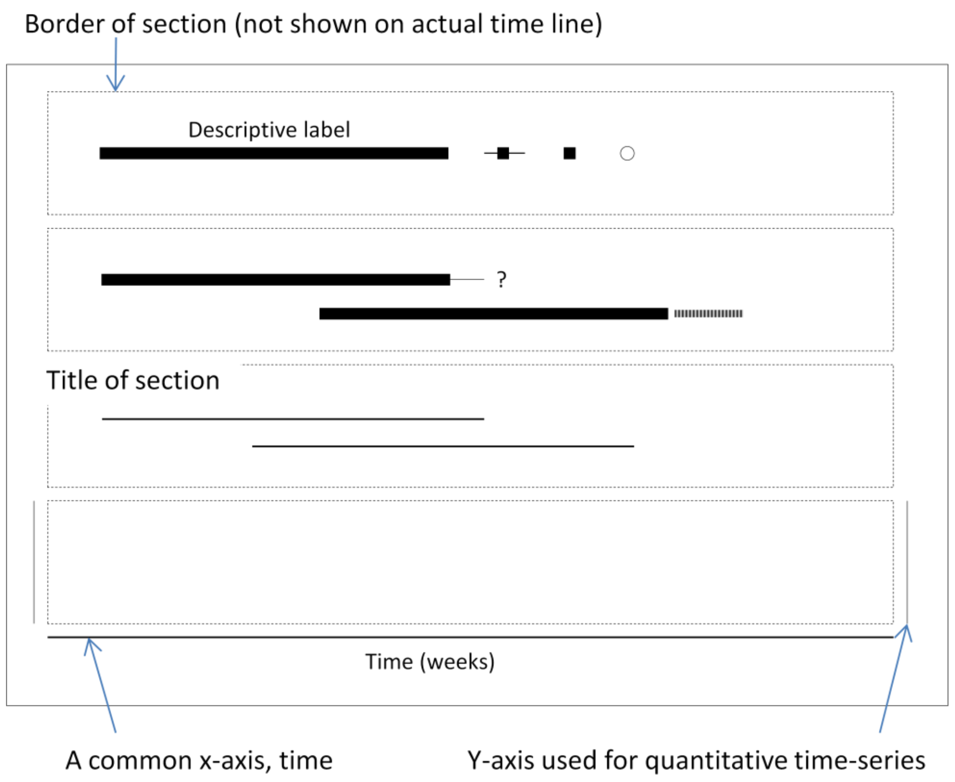
Border of section (not shown on actual time line)
Descriptive label
?
Title of section
Time (weeks)
A common x-axis, time
Y-axis used for quantitative time-series

Figure 1 Simple, annotated example of a multi-dimensional timeline (MDT)

Figure 1 presents a simple, annotated example of an MDT. As already explained, the MDT is organised into sections with all sections on the diagram ordered by a common x-axis shown at the foot of the diagram. For the MDTs in section 5, the x-axis is ordered in weekly units, these units being appropriate to that project. Other MDTs could use other units of time such as seconds, days, months, or years. Each section has a title and a border, although the border is not shown.

The MDTs presented in section 5 were all created manually using Microsoft Excel (to generate the quantitative time-series) and Microsoft PowerPoint (to prepare the visual displays). As a result, quantitative time-series have been presented in the lowest section of an MDT, but this was solely for convenience because it was easier to manually align other non-quantitative information (such as phases and events) with the quantitative time-series, rather than the other way around. Where a section presents multiple quantitative time-series, the y-axis can be used to represent more than one metric, as we show with the workload and capability section in the MDT presented in Figure 2.

All objects in the diagram are accompanied by descriptive labels. Uncertainty over the start or completion of an interval of time or the occurrence of an event are typically illustrated with question mark (i.e. '?') or a thinner line preceding or succeeding a thicker line. Events are represented by filled squares or unfilled circles. Shaded thick lines also indicate overrun.

In principle, it is good design to maintain a consistent notation within a MDT. The use of different notations however helps to emphasise the *principles* of MDTs (i.e. to represent change over time for a variety of types of data and entity) as distinct from the presentational *style* used in any particular MDT.

4 A preliminary appraisal of the multi-dimensional timeline technique

4.1 Research questions

Our general research question for this paper is:

GRQ1 How can multi-dimensional timelines be used to concisely represent the behaviour of a set of project-level attributes as they change over time?

We decompose this research question into four more specific questions viz.

SRQ1 To what degree can MDTs accommodate different types of data?

SRQ2 To what degree can MDTs represent different types of change?

SRQ3 To what degree can MDTs accommodate different conceptualisations?

SRQ4 What are the requirements and limitations of multi-dimensional timelines?

In previous work [49] we identified and collected information on several projects at IBM Hursley Park. For this paper, we use examples from one of the projects, Project B. We use the reference *Project B* to remain consistent with previous work reported on this project e.g. [49-52]. Our previous work has used visualisations to present information on software project behaviour and, specifically, software project schedules. In the current paper, we formalise for the first time those visualisations as multi-dimensional timelines.

4.2 An overview to 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 product family [53, 54] operate on mid-range machines and workstations. The release preceding Project B introduced new transaction logging functionality that required specific hardware to operate; hardware that was not commonly used by customers.

Decision-makers in the product family 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.* [55]), 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 the preceding release, and some functionality that was planned to be delivered in the release that would succeed Project B.

The product family decision-makers also 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 IBM 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 least 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.

4.3 The volume of data, and its diversity

Table 2 Summary of the range and depth of evidence collected for the project

Type	Evidence	Project B
Naturally-occurring evidence	Meeting minutes, of which:	51
	- Project status meetings	49
	- Senior management meetings	1
	- Project review (post-mortem)	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
Research-'generated' evidence	Interviews	8
	Researchers records of status meetings	2
	Feedback workshop questionnaires	1
	Total number of 'documents'	73

Table 2 indicates the range and depth of evidence collected, and analysis conducted on that evidence, for Project B. A relatively large number of documents were collected and much of this documentation was naturally generated by the project itself. The naturally occurring evidence was supplemented by the conduct of interviews and a feedback workshop following the completion of the project.

The primary source of evidence used in the analyses was the minutes of project status meetings. Project B held meetings attended by representatives of all the functional areas in the project. The status meetings were the highest-level meetings within the respective projects, occurred regularly (typically weekly or fortnightly), were typically attended by representatives from functional areas important to the given project, and are a naturally occurring phenomenon (so that the researcher is not intruding on the 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 (e.g. for political reasons, a discussion at the meeting may not be reported in the minutes), and such meetings are unlikely to discuss all the issues occurring within the project at the time of 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 information appears rich enough to provide substantive, longitudinal insights into progress in software development. Furthermore, the minutes provide detail that is unlikely to be collected from other sources of data and that can also indicate simple causal connections between events.

The feedback session was conducted approximately one year after the completion of the project. The Project Leader and Project Assistant were present at the session. The session took the form of exploring the study's findings with the Project Leader and the Assistant, so as to validate and clarify the findings. Van Genuchten [13] adopted a similar approach in his study of software project schedules. More recently, Applicability Checks (e.g. [56] referring to [57]) have been proposed as a method for checking the relevance of a scientific study for practice. The feedback session therefore provides one method of validation of the findings from this investigation. The feedback session also provided additional information to help clarify and extend the analysis.

4.4 Definitions

In preparation for the examples given in section 5, we define three terms here: project-level attributes, project workload and project capability.

The definition of project-level attributes rests on an appropriate definition of a project. There are a wide variety of definitions of projects (see [58] for one review) of which a traditional definition is:

> An endeavour in which human, material and financial resources are organized in a novel way, to undertake a unique scope of work, of given specification, within constraints of cost and time, so as to achieve beneficial change defined by quantitative and qualitative objectives. ([59], quoted in [58])

As noted earlier, Curtis et al. [14] distinguished between five behavioural layers in software design, of which one was the project, but did not provide explicit definitions of these layers. Because projects are abstract entities it is difficult to make clean distinctions between the project and an activity or entity within the project. Nevertheless, project-level *attributes* are attributes of the *entity* project (cf. [60]), examples being: project duration, project schedule, and project budget. Some project-level attributes are composites of a number of more specific attributes e.g. a project's schedule potentially includes information on phases and milestones *of the* project. Inevitably, some of these attributes may be aggregates of within-project attributes, or may be specified by more senior parts of an organisation e.g. an acceptable project duration or overall budget.

Workload is represented in the MDTs in terms of features and design changes. (In Figure 2, 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" ([61]; 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) in contrast to the actual number that occurred on Project B.

Project capability is broadly defined as the ability to complete project workload at a given time in the project. The concept of capability incorporates concepts of productivity and resource, and we use the term capability rather than effort to imply that a project's ability to complete work is not just a function of the number of staff on a project, or the amount of effort available.

For measurement purposes, capability is represented in the figures 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 2. 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 2 to indicate the actual weekly staff levels for his project.

It is clear from Figure 2 that there are some slight increases in staff, but rather than adding people to the project (*cf*. Brook's Law [62]), 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 [63] observation that designers are reassigned to higher priority projects, in this instance remaining with a project rather than being assigned to another project.

5 Examples of the timelines

Figure 2 presents an MDT on the planned and actual schedule, workload and capability of Project B. It is clear from the figure that a considerable amount of change occurred during the project and at different periods of time in the project. For example:

1. The actual phases of work did not progress as planned, with phases completing later than planned and phases progressing concurrently in contrast to the originally planned sequential phases.
2. There was an increase in actual effort toward the end of the project
3. There were considerable increases 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). 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 was 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.) Some design changes are accepted after the apparent actual completion of the design/code phase! This suggests that the design/code phase may actually progress for longer than represented in Figure 2. 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*. [63]) and that the plan does not represent these multiple phases accurately

The figure also indicates uncertainty, for example in terms of when phases start and complete and when decisions were actually made.

A variety of different types of data are presented in the figure:

1. Quantitative time series e.g. design changes, staff levels.
2. Events e.g. decisions.
3. States e.g. phases.
4. Qualitative information, presented as an event with a label e.g. "Decision to proceed with schedule & NOT to slip the GA [general availability] date". (emphasis in original meeting minutes)

In addition to the changes 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 and Madnick [10]) argues that reports of actual progress often simply reflect the planned progress because a more accurate assessment of actual progress is not possible.

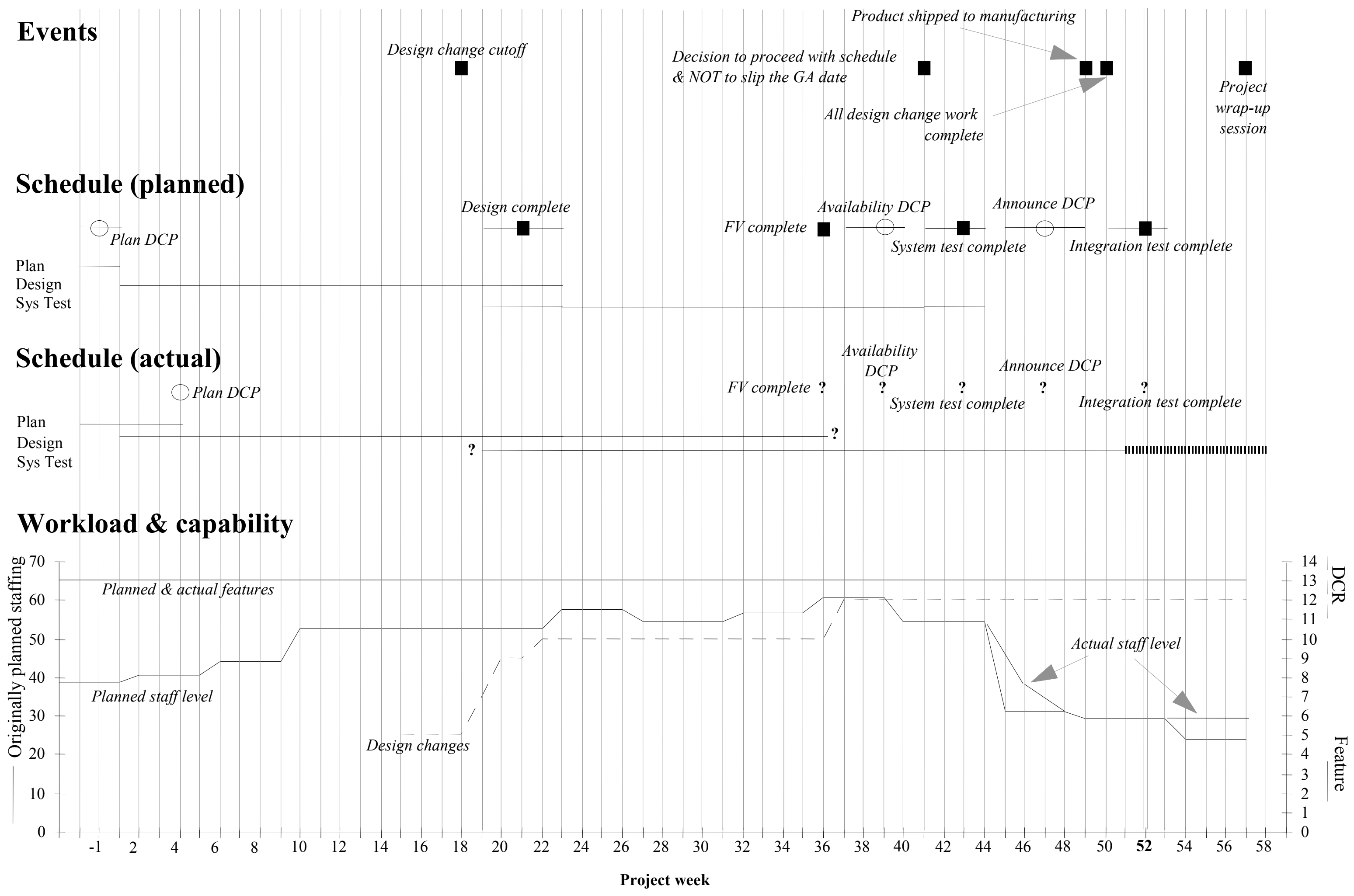

Figure 2 Planned and actual schedule, workload and effort on Project B

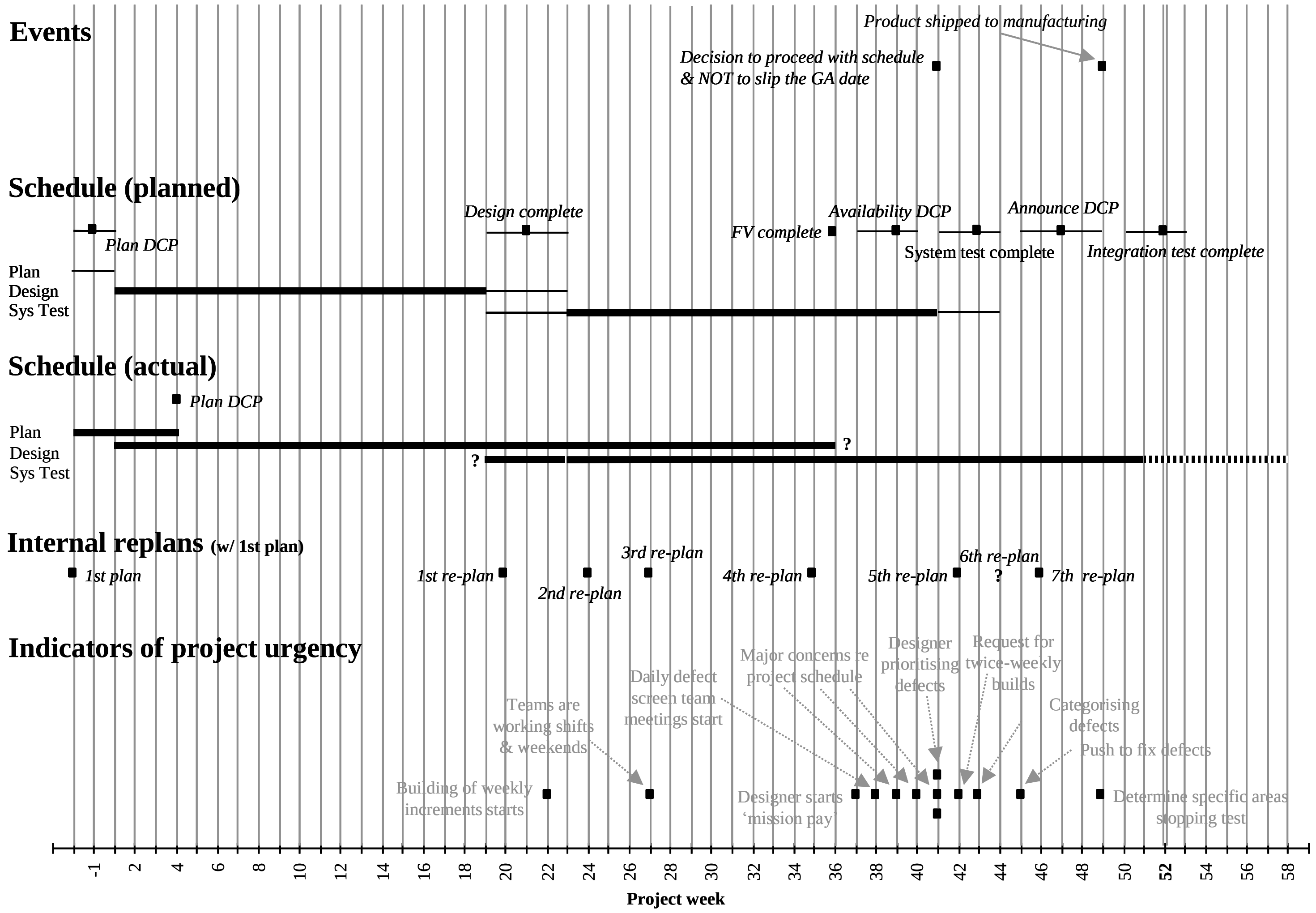

Figure 3 Internal re-plans and indicators of project activity for the project

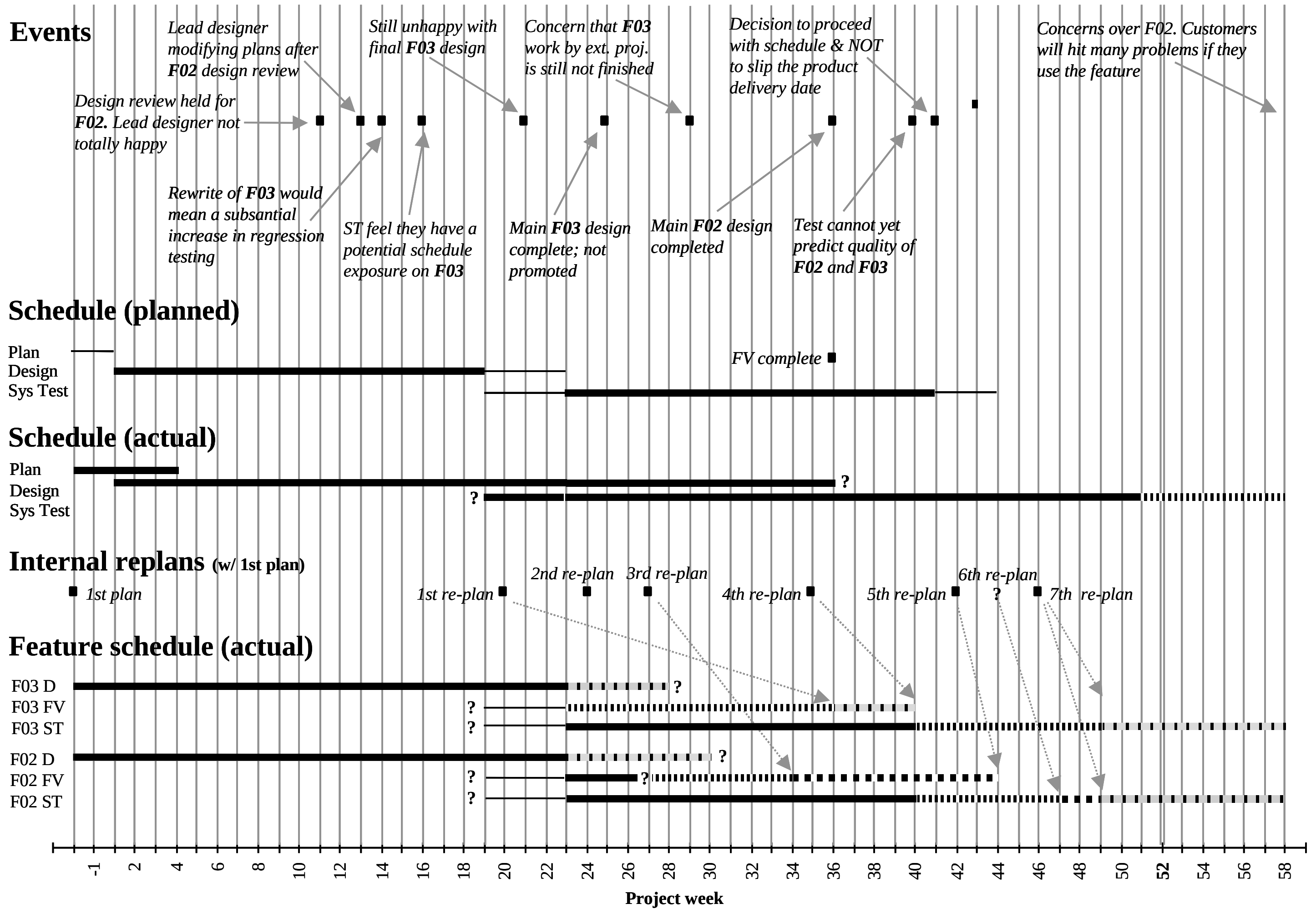

Figure 4 Actual feature schedule for two features of the project

Figure 3 presents an MDT on the planned and actual project schedule, together with the occurrence of internal re-plans, indicators of project urgency and other events. The internal re-plans and indicators of project urgency are the most interesting aspects of this timeline, because they further indicate changes on the project, the frequency of those changes, and when the changes occurred. All of the indicators of project urgency actually comprise qualitative information represented as a labelled event on the timeline.

The internal re-plans are changes to the project plan that do not require senior management approval. Of the seven re-plans on Project B, 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, a change to the plan every month. (The exact week when the sixth re-plan occurs is not known, but this does not affect the calculation of the average because we know when the final re-plan occurs.)

Of the 13 indicators of project urgency, 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.

Figure 2 illustrates the planned and actual behaviour of Project B at a project level. By contrast, Figure 4 compares the planned schedule at a project level with the actual schedule for two features being delivered in the project, as well as related events and project re-plans. The feature schedules are not strictly at the project level, but the MDT shows how project level attributes can be affected by within-project behaviour. The two features (F02 and F03) were chosen because, of all the features and design changes in the project, these two features are referred to most often in the minutes. The frequency of reference in the minutes is presumably because these two features were discussed most frequently in the meetings and the frequency of discussion was presumably because these two features were the most problematic features on the project.

Of seven re-plans on Project B, six of these relate to re-scheduling the phases of features F02 and F03. (The other re-plan concerns the re-scheduling of feature F07.) 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.

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.

6 Discussion

6.1 The research questions

In section 4 we presented a general research question and four specific research questions as a basis for the preliminary appraisal of MDTs. In this section, we consider how the examples presented in section 5 ‘answer’ these research questions. We begin with the specific research questions and then consider the general research question.

SRQ1 To what degree can MDTs accommodate different types of data?

The example MDTs provided in section 5 indicate that MDTs can accommodate at least the following types of data: events, such as decisions; states, such as phases; single and multi-variable quantitative time series; qualitative data, in terms of events with labels; and coded qualitative data i.e. data that has been categorised according to some classification system. Representing qualitative data is more challenging and we return to this point in the discussion of SRQ4.

SRQ2 To what degree can MDTs represent different types of change?

The examples provided indicate that MDTs can represent at least the types of changes relating to the types of data recognised for SRQ1. In addition, the MDTs can be used to infer other information: the point in time at which a change occurs, such as a re-plan; the frequency of a recurring change, such as the number of re-plans; and the period of time over which a change occurs. Through the use of multiple sections in an MDT, different types of change can be represented concurrently e.g. internal re-plans and changes in the number of design changes.

SRQ3 To what degree can MDTs accommodate different conceptualisations?

The structure of MDTs is such that provided a conceptualisation (such as planned or actual schedule, workload, effort, defects) can be represented in terms of events occurring along a temporal dimension then MDTs would appear to be able to accommodate them.

SRQ4 What are the requirements and limitations of multi-dimensional timelines?

The main requirement of MDTs is that the information to be represented (such as the type of data or a change) must be located in time. Also, an MDT can be understood as a geometrical structure consisting primarily of points, lines and planes; so beyond the requirement that information should be locatable in time, it is also necessary for the information to be representable in terms of points (e.g. an event like a decision), lines (e.g. a state like a phase) and planes (e.g. the 'space' for a conceptualisation, such as the y-axis on a quantitative time series or the space to record phases of a planned schedule). There are also of course a variety of labels attached to these geometrical structures and, in the examples presented in section 5, there is at least one type of data where the requirements of an MDT that have just been stated are 'stretched' i.e. when an MDT represents complex qualitative information using labels, such as the series of events presented in Figure 4. In principle, this relatively complex quantitative information could be presented on an MDT by coding the qualitative statement into a set of categories and then representing each of these categories on the MDT, perhaps in their own section of the MDT.

There are limitations to MDTs: they are currently unable to accommodate atemporal information; the amount of information that can be represented on an MDT is limited by the size of the display e.g. an A4 piece of paper; and MDTs do not currently support the representation of causal information, although Figure 4 includes arrows representing the relationship between the internal re-plans and the features that those re-plans concern.

GRQ1 How can MDTs be used to concisely represent the behaviour of a set of project-level attributes as they change over time?

MDTs would appear to be able to represent project-level attributes as they change over time, provided these attributes, and the empirical data about them, can be located in time and can be represented in terms of the simple geometrical structures of points, lines and planes. Changes in attributes are documented at the point in time at which the change occurs. Each attribute of the project would of course need to be represented on the MDT, and there are a number of ways in which that can be done: as a quantitative time series, as an event, as an event with label containing complex qualitative information, or as a schedule.

6.2 Further evaluation of the multi-dimensional timeline technique

A number of papers (e.g. [22, 64-66]) have examined the degree to which researchers evaluate their own technologies. The consensus from these papers was that insufficient evaluation was being conducted by researchers of the software technologies that these researchers had developed. We have proposed the MDT technique, but also recognise that there is a need for further evaluation of this technique. We have found that the MDTs were useful for communicating the results of two case studies at IBM Hursley Park to the respective project managers and project assistants, as well as to a small number of other stakeholders at IBM Hursley Park. These experiences do not however constitute a formal evaluation.

A formal empirical evaluation would be difficult to design and conduct because the evaluator would need to isolate a number of different sets of factors, for example: the underlying conceptual structure of the MDTs, the graphic design of the visualisation itself, the phenomena being represented (e.g. software projects), the quality and nature of the data available about that phenomena (e.g. qualitative and quantitative data, missing data, etc.), the ability of the reader of the visualisation to understand the visualisation as a visualisation, and the ability of the reader to reason about the phenomena being represented in the visualisation.

As a step toward a more formal evaluation in the future, we briefly consider: a review of software tools and techniques for eventvisualisation by Chung et al. [67], and heuristics for the design of information visualizations by Baldonado et al. [68].

According to Chung et al. [67], the temporal and spatial dimensions are the primary focus for many event visualization techniques implemented in software tools. Chung et al. reviewed 24 techniques and tools that have been produced between 1991 and 2003. These tools have been used in a wide variety of applications, including: medicine, law enforcement, business and news, computer mediated communication, environmental studies and entertainment. Many of these software tools support interactive exploration of the temporal and spatial dimensions, and this interactive exploration can increase the speed and accuracy of understanding on the part of the user. There are few evaluations of such tools however, and Chung et al. report only six examples. Given the number of techniques and tools, and the range of their application, event visualisations that support a temporal dimension appear to be valuable for communicating information about phenomena as well as reasoning about those phenomena.

In the field of information visualisation, Baldonado et al. [68] have derived a set of eight heuristics for the use of multiple views in information visualizations. These heuristics were derived from the analysis of existing systems, the authors' own experiences of designing systems, and from participation in discussion at the CHI'98 workshop on information exploration environments. The heuristics are presented here in Table 3. According to Baldonado et al., the first four heuristics (diversity, complementarity, parsimony, and decomposition) provide guidance on the selection of multiple views. These heuristics would therefore apply to the choice and content of sections in an MDT. The last four heuristics (space/time resource optimization, self-evidence, consistency, and attention management) provide guidance on the actual presentation of the visualization. These heuristics would therefore apply to the graphical presentation of the content of sections of any MDT to the reader.

Baldonado et al. [68] also discussed the benefits and costs of each heuristic to the reader of the information visualization as well as to the visualization designer. Further details on these issues can be found in Baldonado et al.'s paper [68].

Table 3 Heuristics for design of information visualizations [68]

Identifier	Heuristic
Heuristics for selecting multiple views	
Diversity	Use multiple views when there is a diversity of attributes, models, user profiles, levels of abstraction, or genres.
Complementarity.	Use multiple views when different views bring out correlations and/or disparities
Decomposition	Partition complex data into multiple views to create manageable chunks and to provide insight into the interaction among different dimensions.
Parsimony	Use multiple views minimally
Presentation and interaction of visualization	
Space / Time Resource Optimization	Balance the spatial and temporal costs of presenting multiple views with the spatial and temporal benefits of using the views.
Self-Evidence	Use perceptual cues to make relationships among multiple views more apparent to the user.
Consistency	Make the interfaces for multiple views consistent, and make the states of multiple views consistent.
Attention Management	Use perceptual techniques to focus the user's attention on the right view at the right time.

Given the above discussion, some of the issues around the evaluation of MDTs requiring further attention are:

1. To what degree can the MDT technique provide a representation of a project that is more concise than, for example, a narrative or a time series? With further development of the time lines (e.g. a more sophisticated notation), could the time line approach make detailed narrative descriptions either unnecessary or at least less necessary?
2. To what degree can the MDT technique provide a representation of a project that effectively supports the identification and evaluation of causal explanations in a project?
3. To what degree can the MDT technique represent different kinds of software engineering phenomena and different kinds of evidence? As suggested by Figure 1, two fundamental requirements of the time line approach are that entities and attributes of a phenomenon of interest can be represented as objects in time (events) relative to each other, and that these objects can be represented using basic geometrical objects (e.g. lines, points, possibly areas). Aside from these two requirements, the MDT technique appears to be very flexible over the kinds of phenomenon that can be represented; but these assumptions require further investigation and evaluation.

6.3 Further research

Further research in the MDTs is required in at least the following areas:

- Further develop the notation so that it can provide a richer representation of project-level behaviour.
- Develop a conceptual framework of events and MDTs.
- Better understand the limits of what MDTs can and cannot represent.

- Better understand the layers of abstraction appropriate for MDTs and how these layers can be related to each other (perhaps similar to the concepts of the levels of a Data Flow Diagram).
- Develop guidelines on when the MDT technique should and shouldn't be used, and how MDTs can be integrated with other kinds of representation.
- Determine the evidence most appropriate for these kinds of descriptions.
- Better understand how to integrate project-level attributes with attributes of processes within the project and 'above' the project.
- Provide a software tool for generating and manipulating MDTs[1].

6.4 Conclusion

Little attention has been directed at modelling temporal aspects of the project-level attributes of software projects. To model these aspects, we propose the use of multi-dimensional timelines (MDTs) that combine quantitative and non-quantitative data to represent a range of project-related attributes relative to a common time axis. We demonstrate the application of MDTs with examples from a longitudinal study of a software project, Project B, at IBM Hursley Park. The examples show how the MDTs can represent qualitative data as events over time, as well as schedule information (actual and planned) and quantitative time series all on the same visualisation. MDTs seem to be capable of accommodating different types of data, representing different types of change, and accommodating different conceptualisations. In general, MDTs would appear to be capable of representing a variety of types of information provided that information and the empirical data about them can be located in time and can be represented in terms of the simple geometrical structures of points, lines and planes. We also briefly consider the range of visualisation techniques and tools being developed in the information visualisation field as an indicator of the communicative and analytic benefits to visualising time; and we also briefly consider heuristics for the design of information visualisations. Finally, we identify a range of further work required on the proposed MDTs, in terms of the further development and evaluation of the technique.

Acknowledgements

I would like to thank IBM Hursley Park and all of the project members of Project B for allowing me to study the project and report the results of that study. In particular, I thank Paul Gibson and John Allan for their support at IBM Hursley Park. I would also like to thank Professor Martin Shepperd for supervising my doctoral studies, which forms the basis for the work reported here. Professor Per Runeson and Dr. Sarah Beecham provided very constructive comments on an earlier draft of this paper. I thank also the anonymous reviewers for their insightful and constructive comments on earlier versions of this paper.

References

[1] The Standish Group, "Chaos Report," 1994.
[2] The Standish Group, "Chaos: A Recipe for Success," 1999.
[3] The Standish Group, *Extreme Chaos*, 2001.
[4] The Standish Group, "Latest Standish Group Chaos Report Shows Project Success Rates Have Improved by 50%," 2003.
[5] The Standish Group, "2004 Third Quarter Research Report," 2004.
[6] C. Jones, "Project Management Tools and Software Failures and Successes," *Crosstalk*, pp. 13-17, 1998.
[7] K. R. Linberg, "Software developer perceptions about software project failure: a case study," *The Journal of Systems and Software*, vol. 49, pp. 177-192, 1999.
[8] J. D. Procaccino, J. Verner, S. P. Overmyer, and M. E. Darter, "Case study: factors for early prediction of software project success," *Information and Software Technology*, vol. 44, pp. 53-62, 2002.
[9] C. B. Seaman, "Qualitative Methods In Empirical Studies Of Software Engineering," *IEEE Transactions on Software Engineering*, vol. 25, pp. 557-572, 1999.
[10] T. K. Abdel-Hamid and S. E. Madnick, "Lessons learned from modeling the dynamics of software development," *Communications of the ACM*, vol. 32, pp. 1426-1438, 1989.
[11] J. S. Reel, "Critical Success Factors In Software Projects," *IEEE Software*, pp. 18-23, 1999.
[12] J. D. Procaccino, J. M. Verner, K. M. Shelfer, and D. Gefen, "What do software practitioners really think about project success: an exploratory study," *Journal of Systems and Software*, vol. 78, pp. 194-203, 2005.

[1] We are investigating the use of the Processing language and environment; visit www.processing.org

[13] M. van Genuchten, "Why is software late? An empirical study of reasons for delay in software development," *IEEE Transactions on Software Engineering*, vol. 17, pp. 582-590, 1991.

[14] B. Curtis, H. Krasner, and N. Iscoe, "A field study of the software design process for large systems," *Communications of the ACM*, vol. 31, pp. 1268-1287, 1988.

[15] T. Rodden, V. King, J. Hughes, and I. Sommerville, "Process modelling and development practice," presented at Third European Workshop on Software Process Technology (EWSPT'94), Villard de Lans, France, 1994.

[16] P. E. Waterson, C. W. Clegg, and C. M. Axtell, "The dynamics of work organisation, knowledge, and technology during software development," *International Journal of Human-Computer Studies*, vol. 46, pp. 79-101, 1997.

[17] W. Hesse, U. Bittner, and J. Schnath, "Results From The IPAS Project: Influences Of Methods And Tools, Quality Requirements And Project Management On The Work Situation Of Software Developers," *Annual Review of Automatic Programming*, vol. 16, pp. 179-183, 1992.

[18] B. Kitchenham, G. Travassos, A. von Mayrhauser, F. Niessink, N. F. Schneidewind, J. Singer, S. Takada, R. Vehvilainen, and H. Yang, "Toward an ontology of software maintenance," *Journal of Software Maintenance: Research and Practice*, vol. 11, pp. 365-389, 1999.

[19] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, pp. 131–164, 2009.

[20] B. Kitchenham, S. L. Pfleeger, L. Pickard, P. Jones, D. C. Hoaglin, and J. Rosenberg, "Preliminary guidelines for empirical research in software engineering," *IEEE Transactions on Software Engineering*, vol. 28, pp. 721-734, 2002.

[21] B. Kitchenham, L. Pickard, and S. L. Pfleeger, "Case studies for method and tool evaluation," *IEEE Software*, vol. 12, pp. 52-62, 1995.

[22] N. Fenton, S. L. Pfleeger, and R. L. Glass, "Science and substance: a challenge to software engineers," *IEEE Software*, vol. 11, pp. 86-95, 1994.

[23] R. K. Yin, *Case Study Research: Design and Methods*, 3rd edition ed: SAGE Publications, 2003.

[24] M. G. Bradac, D. E. Perry, and L. G. Votta, "Prototyping A Process Monitoring Experiment," *IEEE Transactions on Software Engineering*, vol. 20, pp. 774-784, 1994.

[25] M. G. Bradac, D. E. Perry, and L. G. Votta, "Prototyping A Process Monitoring Experiment," presented at 15th International Conference on Software Engineering, 1993.

[26] D. B. Walz, J. J. Elam, and B. Curtis, "Inside A Software Design Team: Knowledge Acquisition, Sharing, And Integration," *Communications of the ACM*, vol. 36, pp. 63-77, 1993.

[27] R. Guindon, "Designing the design process: Exploiting opportunistic thought," *Human-Computer Interaction*, vol. 5, pp. 305-344, 1990.

[28] R. Guindon, "Knowledge exploited by experts during software system design," *International Journal of Man-Machine Studies*, vol. 33, pp. 279-304, 1990.

[29] S. P. Davies, "Characterizing The Program Design Activity: Neither Strictly Top-Down Nor Globally Opportunistic," *10*, vol. 3, 1991.

[30] W. Visser, "Organisation Of Design Activities: Opportunistic, With Hierarchical Episodes," *Interacting With Computers*, vol. 6, pp. 235-238, 1994.

[31] J. E. Cook, L. G. Votta, and A. L. Wolf, "Cost-Effective Analysis Of In-Place Software Processes," *IEEE Transactions on Software Engineering*, vol. 24, pp. 650-663, 1998.

[32] J. E. Cook, "Process Discovery and Validation Through Event-Data Analysis," in *Department of Computer Science*. Boulder: University of Colorado, 1996.

[33] J. E. Cook and A. L. Wolf, "Discovering models of software processes from event-based data," Software Engineering Research Laboratory, Department of Computer Science, University of Colorado, Technical Report CU-CS-819-96, November 1996 1996.

[34] B. Curtis, "Three problems overcome with behavioral models of the software development process," presented at 11th International Conference on Software Engineering, May 15-18th, Pittsburgh, Pennsylvania, 1989.

[35] M. Frese and W. Hesse, "The Work Situation In Software-Development - Results of An Empirical Study," *Software Engineering Notes*, vol. 18, pp. A65-A72, 1993.

[36] W. Hesse, "Theory And Practice Of The Software Process - A Field Study And Its Implications for Project Management," presented at 5th European Workshop On Software Process Technology, 1996.

[37] G. M. Olson, J. S. Olson, M. R. Carter, and M. Storrosten, "Small Group Design Meetings: An Analysis Of Collaboration," *Human-Computer Interaction*, vol. 7, pp. 347-374, 1992.

[38] C. B. Seaman and V. R. Basili, "Communication And Organization: An Empirical Study Of Discussion In Inspection Meetings," *IEEE Transactions on Software Engineering*, vol. 24, pp. 559-572, 1998.

[39] J. Singer and T. Lethbridge, "Studying Work Practices To Assist Tool Design In Software Engineering," presented at International Workshop on Program Comprehension, 1998.

[40] J. Singer, T. Lethbridge, N. Vinson, and N. Anquetil, "An examination of software engineering work practices," presented at Centre for Advanced Studies Conference (CASCON'97), Toronto, November, 1997.

[41] I. Sommerville and S. Monk, "Supporting informality in the software process," presented at Third European Workshop on Software Process Technology (EWSPT'94), Villard de Lans, France, 1994.

[42] I. Sommerville and T. Rodden, "Understanding the software process as a social process," presented at Second European Workshop on Software Process Technology (EWSPT'92), Trondheim, Norway, 1992.

[43] I. Sommerville and T. Rodden, "Human, social and organisational influences on the software process," Lancaster University, Technical Report CSEG/2/1995, 1995.

[44] R. van Solingen, E. Berghout, and F. van Latum, "Interrupts: Just A Minute Never Is," *IEEE Software*, vol. 15, pp. 97-103, 1998.

[45] W. Heijstek and M. R. V. Chaudron, "Evaluating RUP Software Development Processes Through Visualization of Effort Distribution," presented at 34th Euromicro Conference Software Engineering and Advanced Applications Year, Parma, Italy, 3rd - 5th September 2008, 2008.

[46] K. Wnuk, B. Regnell, and L. Karlsson, "Visualization of Feature Survival in Platform-Based Embedded Systems Development for Improved Understanding of Scope Dynamics," presented at Requirements Engineering Visualization (REV'08), Barcelona, Catalunya, Spain, 8th - 12th September 2008, 2008.

[47] W. R. Shadish, T. D. Cook, and D. T. Campbell, *Experimental and quasi-experimenbtal designs for generalized causal inference*. Boston: Houghton Mifflin Company, 2002.

[48] M. B. Miles and A. M. Huberman, *Qualitative Data Analysis*, 2nd ed. London: SAGE Publications, 1994.

[49] A. W. Rainer, "An Empirical Investigation of Software Schedule Behaviour," in *Department of Computing*. Bournemouth UK: Bournemouth University, 1999.

[50] A. Rainer and T. Hall, "Identifying the causes of poor progress in software projects," presented at 10th International Symposium on Software Metrics (METRICS'04), 2004.

[51] A. Rainer and M. J. Shepperd, "Investigating software project schedule behaviour," presented at EASE 98 Empirical Assessment and Evaluation in Software Engineering, Keele, UK, March 30 - April 1 1998, 1998.

[52] A. Rainer and M. J. Shepperd, "Re-planning for a successful project," presented at 6th International Software Metrics Symposium (Metrics'99), Boca Raton, Florida, 4th - 6th November, 1999.

[53] J. D. McGregor, S. Jarrad, L. M. Northrop, and K. Pohl, "Initiating Software Product Lines," *IEEE Software*, vol. 19, pp. 24 - 27, 2002.

[54] K. Pohl, G. Böckle, and F. v. d. Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*: Springer Verlag, 2005.

[55] E. Carmel, "Cycle time in packaged software firms," *Journal of Product Innovation Management*, vol. 12, pp. 110-123, 1995.

[56] R. L. Glass, "Making Research More Relevant While Not Diminishing Its Rigor," *IEEE Software*, vol. 26, pp. 96-95, 2009.

[57] M. Rosemann and I. Vessey, "Toward improving the relevance of information systems research to practice: the role of applicability checks," *MIS Quarterly*, vol. 32, pp. 1-22, 2008.

[58] J. R. Turner and R. Müller, "On the nature of the project as a temporary organization," *International Journal of Project Management 21 (2003)* vol. 21, pp. 1–8, 2003.

[59] J. R. Turner, *The handbook of project based management.* . London: McGraw-Hill, 1993.

[60] B. Kitchenham, S. L. Pfleeger, and N. Fenton, "Toward a Framework for Software Measurement Validation," *IEEE Transactions on Software Engineering*, vol. 21, pp. 929-943, 1995.

[61] L. M. Taff, J. W. Borchering, and W. R. Hudgins Jr., "Estimeetings: Development estimates and a front-end process for a large project," *IEEE Transactions on Software Engineering*, vol. 17, pp. 839-849, 1991.

[62] J. Brooks, F.P., *The Mythical Man-Month*, Anniversary Edition ed. Reading, Massachusetts: Addison-Wesley Publishing Company, 1995.

[63] D. E. Perry, N. A. Staudenmayer, and L. G. Votta, "People, organizations, and process improvement," *IEEE Software*, vol. 11, pp. 36-45, 1994.

[64] R. L. Glass, "The software research crisis," *IEEE Software*, vol. 11, pp. 42-47, 1994.

[65] W. F. Tichy, P. Lukowicz, L. Prechelt, and E. A. Heniz, "Experimental Evaluation in Computer Science: A Quantitative Study.," *Journal of Systems Software.*, vol. 28, pp. 9-18, 1995.

[66] M. V. Zelkowitz and D. Wallace, "Experimental validation in software engineering," *Information and Software Technology*, vol. 39, pp. 735-743, 1997.

[67] W. Chung, H. Chen, L. G. Chaboya, C. D. O'Toole, and H. Atabakhsh, "Evaluatingevent visualization: a usability study of COPLINK spatio-temporal visualizer," *International Journal of Human-Computer Studies*, vol. 62, pp. 127-157, 2005.

[68] M. Q. W. Baldonado, A. Woodruff, and A. Kuchinsky, "Guidelines for Using Multiple Views in Information Visualization," presented at Working Conference on Advanced Visual Interfaces (AVI'00), Palermo, Italy 2000.