A Framework for Linking Projects and Project Management Methods

---

A thesis
submitted in partial fulfilment
of the requirements for the Degree
of
Master of Science
in the
University of Canterbury
by
Tony Dale

University of Canterbury
2006

To my parents and to my wife, Anne.
Anne coined the name: Project Mentor.

Abstract

Software development processes such as the Waterfall process and Extreme Programming are Project Management Methods (PMMs) which are well known and widely used. However, conventional Project Management (PM) lacks the process concepts expressed in PMMs, and the connection between PMMs and PM is not much explored in the literature. Our research problem is to make this connection.

We present data models for PM and PMM, in a framework that can articulate the PM–to–PMM relationship, illustrating with simple examples. Tools and visualizations created in terms of our framework can make use of the familiarity, history and context of project management tools, and the prescriptiveness and reactivity of PMMs, and we believe these may assist the management of complicated projects, such as IT projects.

Project Mentor, a prototype Java/XML implementation of the framework semantics, can create and then revise a "PMM–aware" project, conforming to a specified PMM. The PM–to–PMM connection is persistent in project data, and we describe a visualization of the "footsteps" of a PMM in project data that does not rely on the state of a PMM process. The visualization can also be used by Project Mentor, to indicate the state of a PMM.

We test for possible applications of our framework with a case study and survey of some existing project data, and conclude with a description of further work.

Acknowledgments

Firstly, thanks to Dr Neville Churcher, my supervisor, for his ideas, criticism and efforts to make this a better thesis. Dr Warwick Irwin and Dr Brent Martin, my co–supervisor, contributed many ideas, criticisms and insights. Anne Heffernan–Dale, Dr Robert Biddle and Dr Blair McMaster critiqued my draft thesis. Dr Peter Heffernan gave helpful advice for my survey in chapter 9.

Table of Contents

List of Figures

List of Tables

Abbreviations

ACWP Actual Cost of Work Performed

API Application Program Interface

BCWP Budgeted Cost of Work Performed

BCWS Budgeted Cost of Work Scheduled

BPEL4WS Business Process Execution Language for Web Services

BPML Business Process Markup Language

CORBA Common Object Request Broker Architecture

DOM Document Object Model

DTD Document Type Definition

IDE Integrated Development Environment

INCIS Integrated National Crime Information System

JAXB Java Architecture for XML Binding

JDOM Java Document Object Model

LDAP Lightweight Directory Access Protocol

MILOS Minimally Invasive Long-term Organizational Support

MSPDI Microsoft Office Project 2003 XML Data Interchange Schema

NZS New Zealand Standard

PERT Program Evaluation and Review Technique

PM Project Management

PMM Project Management Method

PMXML Project Management XML Schema

RUP Rational Unified Process

SAX Simple API for XML

SVG Scalable Vector Graphics

TPMM Tony's Project Management Method schema

UML Unified Modelling Language

URI Uniform Resource Indicator

WBS Work Breakdown Structure

WFMS Work Flow Management System

XML Extensible Markup Language

XPDL XML Process Definition Language

XSD XML Schema Definition

XSL Extensible Style sheet Language

XSLT XSL Transform

Chapter I

Introduction

1.1 Project Complexity

Large–scale project management has been practised for centuries, to the extent that Burbridge [11] writes

> one hallmark of civilization is the ability to engage in group activities for the execution of major projects, be they tombs and temples or manned flights into space.

With the constant increase of human knowledge, the complexity of projects has increased as well. Some of the most complicated projects undertaken are those involving Information Technology (IT).

Unfortunately, projects in fields such as IT suffer from well–known problems [10][33][14] [70][44][38]. Project failure is not simple to define (see, eg: [38] p. 13 for a discussion), but notwithstanding this, the Standish Group reported in 2001 that only 28% of IT projects were completed on time and within budget [82], although this was an increase from the 16% of 1994 [81]. The INCIS disaster [79][88][19] is an exemplary large IT project failure, and it had a disastrous effect on NZ Police in the late 1990s. The INCIS business case [16] proposed that savings and income from INCIS would allow Police to cut budgets and staff. No such cuts could be made, and other areas of Police operation, such as property maintenance, were neglected in order to pay for the INCIS failure [40].

As we will see in the next few sections, many attempts to improve the success rate of IT projects have centred around providing more powerful tools for managing projects. A central theme of this thesis will be a search for ways to improve these tools.

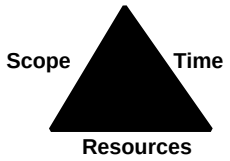


Figure 1.1: *This "project triangle" illustrates the interdependency of the critical parameters of a project: time, cost and scope (the goals of a project).*

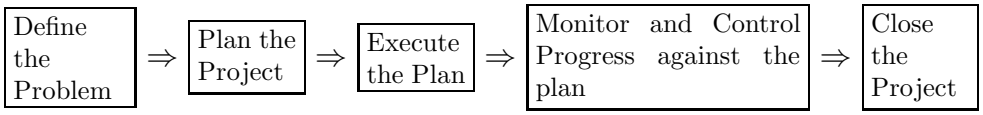


Figure 1.2: *The five steps of managing a project as listed, for example, by (Lewis [49], p7).*

1.2 Project Management (PM)

A crucial change in the management of projects from the 19th century onwards has been the increasing importance of time and cost in project management. Fayol's [31] five functions of management (plan, organize, command, coordinate, and control) form the basis of modern PM, and Gantt [37] introduced methods for measuring project progress against plans.

Modern project management aims to achieve a project goal in the most efficient way, for reasons such as increasing company profitability or reducing expenditure of public funds. Compared with the "craftsman" culture of historic projects (where, for example, generations of one family might work on a medieval cathedral), modern projects must be completed *on time and within budget*, according to comprehensive, predefined, project plans. A project plan consists of a list of tasks, resources, and allocations of resources to tasks, with the aim of achieving the goal or goals of a project, and the "project triangle" of figure 1.1 illustrates the interdependency of these three parameters of a project. Conventional modern Project Management (PM) is encapsulated in the five steps of figure 1.2 and, as we shall see, in the approach of popular project management tools such as Microsoft Project [55]. Mod-

ern approaches to project management, such as those listed in the Project Management Body of Knowledge [67], build on these five steps.

The basic premise of the PM approach is that a problem can be solved by executing a plan that is formulated in advance. Once a project is started, conventional Project Management is concerned with monitoring the progress of a project against the plan, using various well–known methods to measure and report on the project, such as the Critical Path method [47], Gantt charts [37], PERT [66] and Earned Value Analysis [85]. These are all methods aimed at assisting the project manager to keep the progress of a project as close as possible to the pre–defined plan. When the progress of a project deviates from a plan (often by falling behind predicted progress) the alternative solutions involve altering the parameters of the project triangle of figure 1.1. Either more or different resources can be used to keep the project tracking the plan, or the plan can be changed to fit the actual situation of the project.

When managed according to a predefined plan, complicated projects tend to become difficult to control, as the actual project implementation deviates from the planned implementation. IT projects suffer especially from such problems, because of their great complexity and other reasons such as the use of new and untried information technology.

1.3 Project Management Methods (PMMs)

One reaction to the crisis in IT project failure has been the rise of agile methods [52], the epitome of which is Extreme Programming (XP) [5], which is dominated by its eponymous process, and the motto of XP is "embrace change". XP is the latest in a long series of IT "recipes for success", dating back at least to the Waterfall Method [73]. The Waterfall Method looks something like an abstracted IT project plan (indeed, it is often applied in a one-time-through way), but it contains the idea of re-executing tasks, according to the success or failure of some validation test. Such a reactive system is different to a static project plan, as used in PM. We will coin the phrase "Project Management Methods", or PMMs, to describe these reactive, process–oriented methods.

If we apply a PMM to a problem and record the resulting tasks, resources

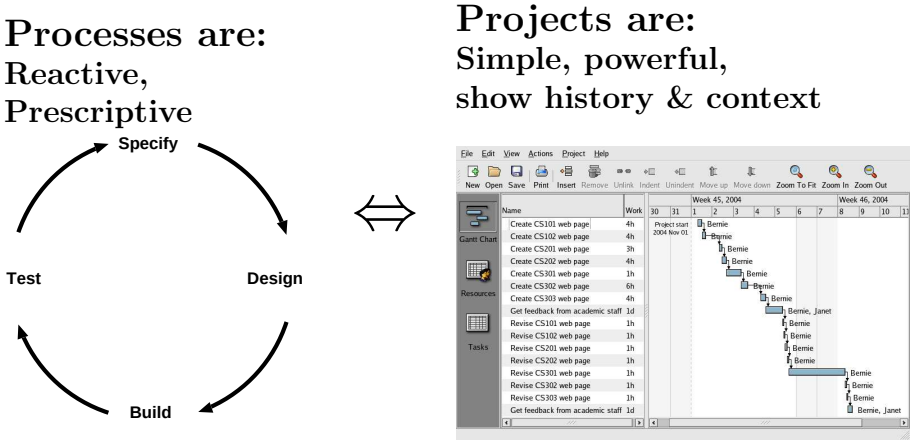


Figure 1.3: *Both processes and projects have advantages that we may be able to combine, by finding a way to link the two domains of knowledge.*

and allocations as a project plan, we end up with a system that looks conceptually like figure 1.3. Relating PMMs to projects may provide assistance from both knowledge domains to project managers, with the familiar concepts and powerful measurement and visualization tools from PM, plus the reactivity of PMMs. PMMs could be used to automatically create and revise projects, and to describe project data. In the next chapter, we will use this basic premise to formulate the research problem that this thesis will address.

1.4 Overview

This thesis expands on a paper that was published at PROFES2005 [18], and it is set out as follows: Chapter 2 describes the research problem and our approach to it. In chapters 3, 4 and 5 we will model the problem domain. We will describe in chapter 6 the data models and semantics of the framework that forms the foundations of our solution. In chapter 7 we will describe three different ways to implement the framework. Chapter 8 describes some applications of the framework to problems that project managers face, and chapter 9 uses a case study and a survey to test the usefulness of our solution to the research problem. We will conclude in chapter 10 with a description of our contribution to knowledge and possible future work.

Chapter II

Research Approach

In this chapter we formulate a research approach to address the issues introduced in chapter 1.

2.1 Research Problem

We have seen in chapter 1 that conventional PM (which we will describe in more detail in chapter 3) attempts to manage tasks and resources as closely as possible to a predefined plan. However, a project plan does not indicate how its particular tasks or resources were created, except perhaps for a textual description. This is because conventional PM has no concept of why a task or resource appears in the plan. PMMs have this descriptive power because they use process concepts to formalise the specialist knowledge which any real project requires; when creating a project plan, for example. A well–defined connection between PMMs and projects may be beneficial because:

- it allows us to describe why a task or resource appears in a project plan, according to the PMM that created the project.
- it allows us to create project plans according to any arbitrary PMM, automatically, and at arbitrary levels of complexity.
- PMMs can be applied to projects in a consistent and explainable way.
- we can make projects more robust by giving them an ability to react to change.
- the project data provides history and context for a PMM process.

- we can find the "footprints" of a PMM in project data, so we can, eg: test conformance of a project to a PMM.

- it affords the possibility of visualizing PMM behaviour in project data.

Some of the potential applications of such a PMM-to-project connection include:

- Automation of project creation and revision may facilitate finer–grained definition and control of a project.

- Many different project scenarios could be analyzed, applying different PMMs to the same goal, for example, or by varying the critical parameters of the project scenario.

- Project metrics could be defined in terms of a PMM. For example: the number of iterations of a code/test loop could be tracked for excessive iterations.

- Project data that deviated from the specification of a PMM could be used to revise the PMM.

We will develop the above ideas in the rest of this thesis.

2.2 Current Approaches and their limitations

In our survey of the literature we found tools that used process–oriented or project–oriented architectures, but not both together, tools that linked PM and PMM with message–passing architectures and project tools that use an embedded PMM.

2.2.1 Process architectures

There are many existing process modelling architectures. For example, the development process architecture of SUKITS [89] consists of a process instance level that is described by a process definition level, and a "technical layer", which is a finer–grained representation of processes.

There are also many different process description languages in existence already (eg: XML languages include BPML[8], XPDL [86], BPEL4WS [7], etc) and work is underway on their interoperability [74][78]. However, these languages are not specifically aimed at PMMs, and for our purposes they neglect such important issues as resource assignment [97][75]. These architectures are also completely process–oriented, not considering PM at all.

2.2.2 Project architectures

We found that popular Project tools (described further in sections 3.2 and 3.3) used an architecture based on tasks, resources and allocations, and this commonality allows many Project tools to exchange data—of which more in later chapters. Tools such as IBM's Rational Project Tracker [63] can view or exchange tasks with the widely–used Microsoft Project tool. Similarly, the Planner[42] project tool can import Microsoft Project XML data. However, such facilities are a PM–to–PM link, and do not add any process concepts to a project.

2.2.3 Connecting PMMs to Projects

Bussler [12] describes two ways to connect between Processes (in the form of Workflow Management Systems or WFMSs) and PM, at the database level, or at the process level. Bussler doesn't develop the idea of a database–level connection, but rather goes on to describe a process–level connection, shown in figure 2.1. APIs between a PMM and a PM tool are used, with message–passing between the PM and PMM software tools to maintain the, largely state–based, connection. Bussler lists six additional requirements for a workflow management system, for instance, to support the message–passing.

The MILOS [50] system implements Bussler's logic–level linkage between PM and PMMs in a 3–tier architecture (figure 2.2). MILOS has a message–based linkage to an augmented Microsoft Project, which it can use to describe PMMs. MS Project is only able to describe the "PM concepts" of table 2.1, but it does have the advantage of being a familiar tool for project managers. More complete PMM description facilities are available from the MILOS

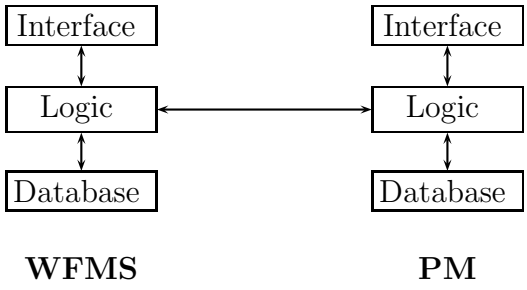


Figure 2.1: *Bussler proposed linking a WFMS (equivalent to our PMMs) to project management at the logic level.*

application itself.

2.2.4 Project Tools with Embedded PMMs

Another way of linking PM and PMM domains is to embed a PMM in a PM tool, so that the tool "acts out" the PMM, and will only allow certain kinds of tasks in any particular circumstance. This is the approach taken by such project management tools as Maven [51], for instance. The embedded PMM of Maven requires that a software project must be "Maven enabled" before the tool can be used on it. Similarly, the PAM Distributed XP tool [91] contains embedded Extreme Programming practices, with support for a widely–distributed team of programmers.

The advantage of such tools is the comprehensive support they provide for users of a particular PMM: Maven automates such practices as unit testing and a code repository, and PAM provides sophisticated facilities to support a geographically separated Extreme Programming team. These tools do not allow an alternative PMM to the one chosen, so that it would be difficult or impossible to use PAM or Maven to support the Waterfall Method, without altering the tools.

2.3 Our Proposed Solution

We propose a data–modelling solution to the research problem. Bussler's logic–level link between PM and PMM is workable, but linking the two do-

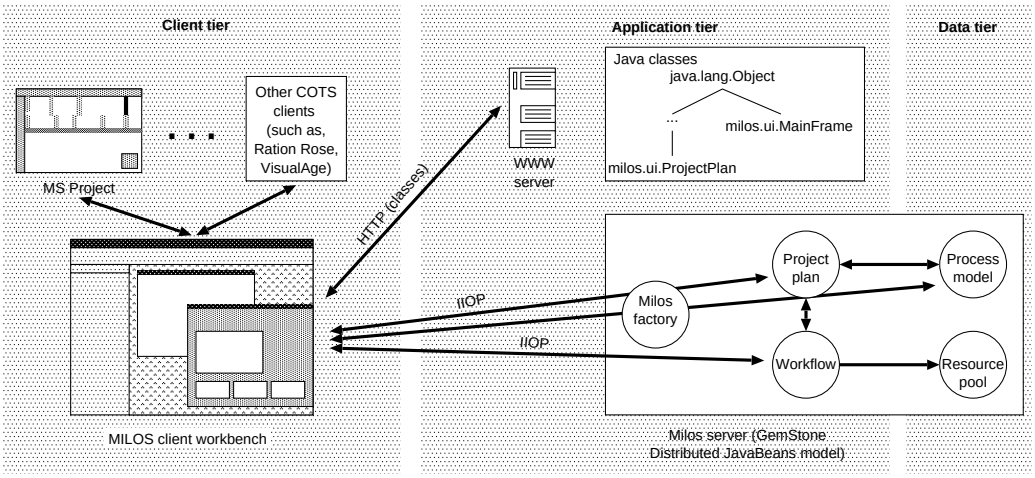


Figure 2.2: *Figure 1 from [50]: the MILOS 3–tier architecture, which uses CORBA [15] for the PM/PMM linkage.*

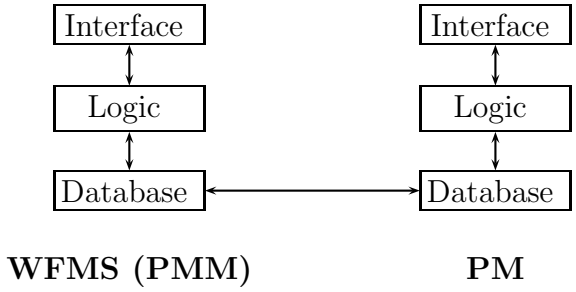


Figure 2.3: *Our proposed PM↔PMM linkage, operating at the database level.*

mains at the database level, as illustrated in figure 2.3, has these advantages:

- The connection between PM and PMM data is persistent, so that the PMM cannot be "lost" from the project data.

- It is possible to integrate extant PM and PMM tools which have no existing connection with, or "knowledge" of, each other.

- The PMM can be used to describe PM data so that, eg: visualization of PM data in terms of a PMM becomes possible.

- Implementations of the connection are not limited to a specific architecture or software tool.

To investigate these possibilities, we will formulate a data model framework that links PM and PMM. There are several database technologies used in PM tools that could be explored for this purpose. For example: Microsoft Project Server is based on Microsoft SQL server. Microsoft supplies example scripts to extract data from the SQL server database of PM data for reporting purposes [56] [54]. Another database technology is XML (eXtensible Markup Language [93][96][9][3]), and there are published XML schemata for PM data; PMXML [68], for instance. Microsoft offers an XSL style sheet for reporting unfinished tasks from Microsoft Project 2002 XML data [53]. To accommodate as many different database technologies as possible, it is desirable to create a framework that is independent of any particular one.

Bussler identifies a number of mapping problems from the PMM domain (that Bussler calls a "Work Flow Management System" or WFMS) to the PM domain, and we reproduce Bussler's list in table 2.1. We will address Bussler's list in the creation of our proposed framework.

2.4 Conclusion

In this chapter we have briefly described current work and then introduced the research problem to be addressed in the rest of this thesis. In the next chapter, we will describe and model the project (PM) domain, and then

WFMS (PMM) concept	**PM Concept**	**Mapping Problems**
Composite & elementary workflow	Task	In PM, elementary and composite tasks are not distinguished by type but by their position relative to each other.
N/A	Tasks have duration	No mapping since duration of tasks is managed in PM only. If a workflow step has a duration attached to it, this can be mapped.
Sequence Conditional Branching Parallel Branching Recursion Loops N/A N/A	End–to–begin N/A Parallel Branching N/A N/A Begin-to-begin End-to-end	PM does not support conditional branching, recursion or loops. However, conditional branching, loops and recursion can be dynamically imitated during execution (see Section 3.4). WFMSs in general do not support begin–to–begin or end–to–end relationships.
Users, groups, roles, etc, together with complex assignment rules.	Resource	PM does know about resources as individuals. No role resolution or more sophisticated resource assignment can be specified. Since all assignments boil down to individual users and only these have capacities, complex rules are not addressed within project management.
N/A	Resources have capacity and load	No mapping problem since capacity and load are managed by PM only. In case the definition of resources within a WFMS includes capacity, it can be mapped to the PM.
Data and Dataflow	N/A	Data are not relevant for scheduling. However, explicit data flow within a workflow type might add dependencies between tasks. In this case additional end–to–begin dependencies have to be introduced where data flow adds dependencies.
Application integration	N/A	PM is not concerned with application integration.

Table 2.1: *Table 1 from [12]: Bussler's mapping of workflow management system (PMM) concepts to PM concepts at the "logic level".*

illustrate the basic commonality of some common PM software tools we examined, by examining their database schemata: Can one PM tool exchange data with another? What are the common features, and can they be summarized in a generic PM model? In later chapters, we analyze how PMMs work, and how they can be represented. After that, we explore the question of how we can link Project and PMM data, and still maintain a strong informational coupling between the two domains, as envisaged in this chapter.

Chapter III

Project Management (PM)

In this chapter we will describe how project management is supported by common software tools. The project data stored by software tools reflect concepts essential to project management such as tasks, resources and allocations. To model these essential features of project data, we will use the common features of data models used by some example software tools, testing our ideas by transforming project data from one model to another. The resultant data model of essential PM concepts will be used in later chapters to formulate our solution to the research problem.

3.1 Projects and Project Management

A project is generally [49][67] realised as a series of tasks that take place over a fixed period of time, in order to achieve a specified, unique, goal. For example: the production of thousands of identical refrigerators in a factory is not a project, but the design of a new model of refrigerator is. Project management is the technique used to keep a project tracking a predefined plan as closely as possible. We shall see in chapter 4 that there are disadvantages to this approach, such as the difficulty of coping with changes to the project plan almost always required in any real project. However, Project Management allows the straightforward production of project budgets and time lines and the visualization of project progress, eg: with a Gantt chart.

The five stages of project management we illustrated in figure 1.2 on page 2 are: defining the problem, planning a project, executing the plan, monitoring and controlling progress against the plan and closing a project. There are many variations on this basic approach to PM: the Project Management Body of Knowledge [67] gives several examples of a "project life

cycle", and for defence systems acquisition [26] there are steps of Concept and Technology Development, System Development and Demonstration (ie: prototyping), Production and Deployment, and ongoing Support. For construction projects, there is Feasibility, Planning and Design, Construction and Turnover and Startup. All of these approaches differ in detail, but are fundamentally similar in that they contain project definition, planning, execution, control and closure activities, that we describe in detail in the next sections.

3.1.1 Defining the Problem

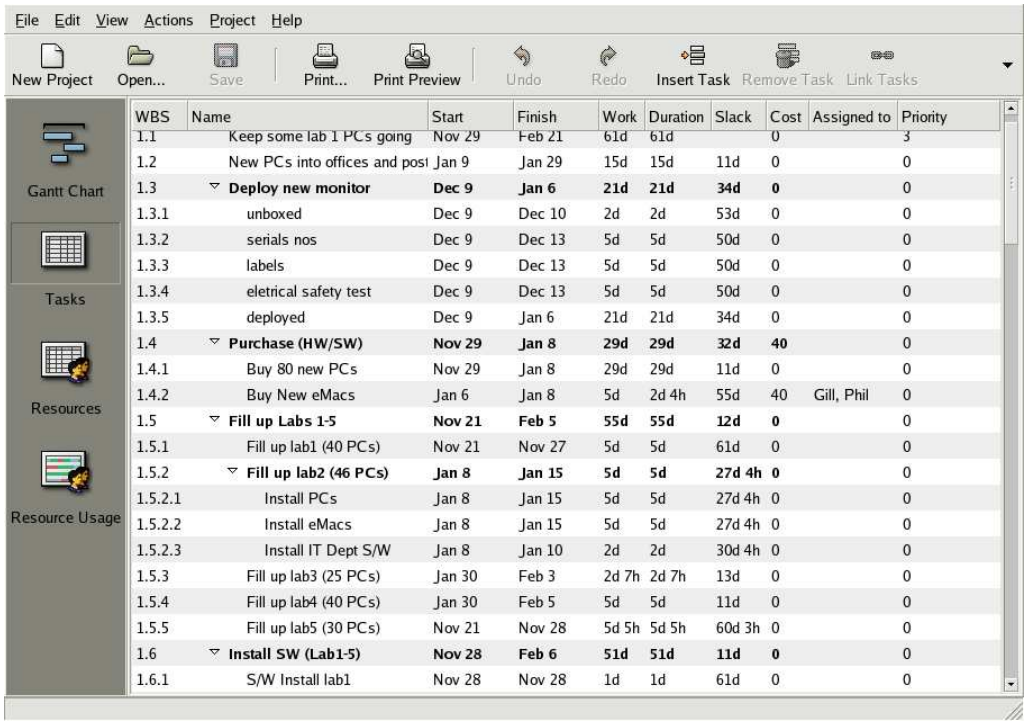
A project is initiated to achieve a specified goal. The initiator of a project, or "Sponsor" [61][79], specifies the goals for the project to achieve (known as the project "scope"), and often provides the resources (as requested for the project) to achieve the goals. A suitable sponsor for a project would be the CEO of a company.

3.1.2 Planning a Project

Having been set a goal to achieve, a project team will plan the tasks required to achieve it. Some of the tasks in the list may have predecessors that set an ordering. The tasks will have estimates of work and required resources, and so once the list of tasks and task ordering has been drawn up, not only will the resource requirements be known, but each resource will have a schedule of tasks it will be allocated to. These tasks, resources and allocations are codified in a project plan, a document consisting of, essentially:

- A list of tasks, which have attributes such as a name, description, percentage complete, start time and estimated work, such as figure 3.1 shows. Tasks may have an ordering set by direct or indirect predecessors, so that one task must finish before another can start[1]. A set of small tasks can be aggregated to form a larger–scale summary task, that derives such properties as its task length from the smaller tasks.

[1] or start before another task can finish, or start at the same time, or finish at the same time



WBS	Name	Start	Finish	Work	Duration	Slack	Cost	Assigned to	Priority
1.1	Keep some lab 1 PCs going	Nov 29	Feb 21	61d	61d		0		3
1.2	New PCs into offices and post	Jan 9	Jan 29	15d	15d	11d	0		0
1.3	**Deploy new monitor**	**Dec 9**	**Jan 6**	**21d**	**21d**	**34d**	**0**		0
1.3.1	unboxed	Dec 9	Dec 10	2d	2d	53d	0		0
1.3.2	serials nos	Dec 9	Dec 13	5d	5d	50d	0		0
1.3.3	labels	Dec 9	Dec 13	5d	5d	50d	0		0
1.3.4	eletrical safety test	Dec 9	Dec 13	5d	5d	50d	0		0
1.3.5	deployed	Dec 9	Jan 6	21d	21d	34d	0		0
1.4	**Purchase (HW/SW)**	**Nov 29**	**Jan 8**	**29d**	**29d**	**32d**	**40**		0
1.4.1	Buy 80 new PCs	Nov 29	Jan 8	29d	29d	11d	0		0
1.4.2	Buy New eMacs	Jan 6	Jan 8	5d	2d 4h	55d	40	Gill, Phil	0
1.5	**Fill up Labs 1-5**	**Nov 21**	**Feb 5**	**55d**	**55d**	**12d**	**0**		0
1.5.1	Fill up lab1 (40 PCs)	Nov 21	Nov 27	5d	5d	61d	0		0
1.5.2	**Fill up lab2 (46 PCs)**	**Jan 8**	**Jan 15**	**5d**	**5d**	**27d 4h**	**0**		0
1.5.2.1	Install PCs	Jan 8	Jan 15	5d	5d	27d 4h	0		0
1.5.2.2	Install eMacs	Jan 8	Jan 15	5d	5d	27d 4h	0		0
1.5.2.3	Install IT Dept S/W	Jan 8	Jan 10	2d	2d	30d 4h	0		0
1.5.3	Fill up lab3 (25 PCs)	Jan 30	Feb 3	2d 7h	2d 7h	13d	0		0
1.5.4	Fill up lab4 (40 PCs)	Jan 30	Feb 5	5d	5d	11d	0		0
1.5.5	Fill up lab5 (30 PCs)	Nov 21	Nov 28	5d 5h	5d 5h	60d 3h	0		0
1.6	**Install SW (Lab1-5)**	**Nov 28**	**Feb 6**	**51d**	**51d**	**11d**	**0**		0
1.6.1	S/W Install lab1	Nov 28	Nov 28	1d	1d	61d	0		0

Figure 3.1: *A list of tasks for a project produced by the Planner[42] project tool. Sub–tasks are indented from their summary tasks, and the cost of tasks is worked out by multiplying the amount of work estimated for the task by the cost of the resources assigned. The costs are not assigned any particular currency, although many project tools can do so.*

- A list of resources, which have name, cost (of the resource) and description attributes.

- The allocations of resources to tasks.

A project plan will contain at least the above concepts, but additions may be made to the project plan, often to facilitate advanced reporting of complicated projects. For example: such schemes as a Work Breakdown Structure (WBS), a hierarchical classification of tasks such as that used by the US military [57], may be used to classify tasks according to their specificity, or type of resource requirements. So–called "milestone" tasks are used to establish a task that must complete, often by a preset deadline.

The project plan may bring to light issues with the project such as a lack of sufficient resources to execute all of the tasks scheduled in a certain time. The plan may require revision to deal with the issue—tasks could be rescheduled or more resources allocated, for example.

3.1.3 Executing the plan

The project plan specifies which tasks are to be executed, and when, and so the project begins with the first task(s) scheduled on the plan: the resources allocated to the task(s) are directed to the activities specified by each task. Subsequent tasks will be executed according to their start time, or other constraints such as predecessor tasks. In this way the project proceeds, as specified by the plan, until all tasks are completed.

Not all the entities in a project need access to the complete plan—resources need only be concerned with the tasks they are allocated to, for instance. Because there is a project plan with tasks and estimated lengths, resources and allocations, it is possible to produce a schedule of task–related activities for each resource, as shown in figure 3.2.

3.1.4 Monitoring and Controlling Progress against the Plan

Project progress is measured by monitoring the progress of each task, as it is executed. Each task in the project plan has a "completeness" attribute that changes from zero up to 100 percent. The rate of completion of tasks

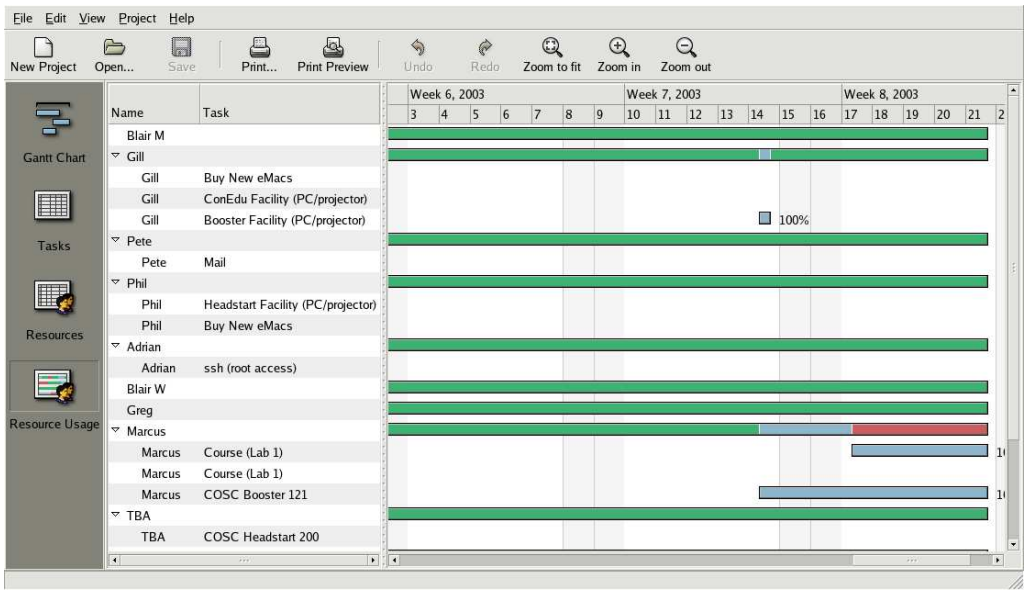


Figure 3.2: *A schedule of resource utilization produced by the Planner project tool.*

is monitored with reference to the estimates in the project plan. Because a Project plan consists of tasks, resources and allocations, it is possible to estimate and measure the time and cost, used and remaining, of a project at any point during its execution. Project progress is measured against time, eg: with a Gantt chart, or cost, using such methods as earned value analysis.

Gantt Charts

Gantt charts ([37], p82) have been used for almost 100 years. Their popularity is due to their powerful features for visualizing the history and progress of a Project. If we refer to the example of figure 3.3, we can see the features of a modern Gantt chart:

A list of tasks down the left hand side of the chart. A modern addition to Henry Gantt's original chart are summary tasks, that aggregate a list of sub–tasks. Summary tasks derive such attributes as task length and start time from the tasks they aggregate.

A time line along the chart. Calendar time progresses from the left to the

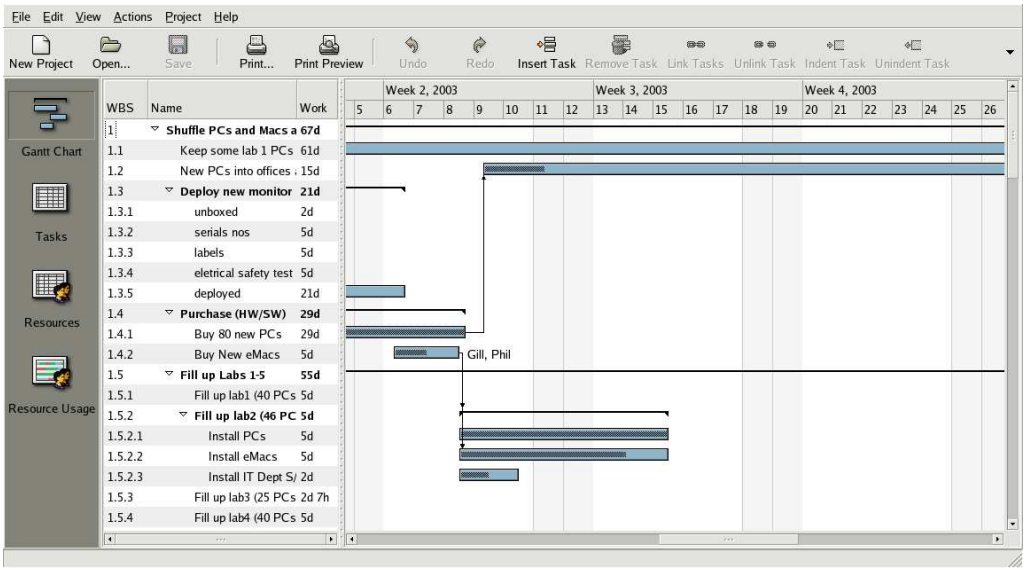


Figure 3.3: *An example Gantt chart from the freeware Planner project management program.*

right of the chart. In western culture, weekends are usually not work times, so they are grayed out by default and tasks are broken across them.

Task bars filled in to indicate the percentage completion. The task bar begins at the time the task is scheduled to begin, and ends at the task finish time. The bar is filled in to indicate an estimate of the task completeness.

Predecessor arrows These arrows indicate the sequence in which tasks must be completed: an arrow is drawn to a task from its required predecessor. In modern projects, predecessors are of four types:

Start–to–start : two tasks must start together.

Start–to–finish : one task must start before the predecessor finishes.

Finish–to–start : one task must start after the predecessor finishes.

Finish–to–finish : two tasks must finish together.

There may be a time lag included in a predecessor. This lag is included in the task schedule as a required delay after a task has finished.

Critical path The Critical Path Method [47][17] identifies the time required to complete a project by adding the durations of the longest sequence of tasks (the "critical path"). Tasks not on the critical path may be delayed as long as they don't impact on the project duration: this is the "slack time". Tasks on the critical path may be highlighted by a software tool (not shown in figure 3.3).

Milestones Milestones are like tasks in that they can be included in a chain of predecessors, but instead of having work associated with them they are a constraint to indicate that all their predecessor tasks must have completed. Milestones might have a fixed date associated, or they might simply be part of a sequence of tasks. Normally the name and/or description of the milestone is used to describe what they indicate (eg: "Design step complete").

Measuring Project Progress with Earned Value Analysis

Earned value analysis as outlined by, for example: Lewis ([49], pp92–4) measures project progress by accounting for the cost of a project in either dollar value or hours of work. Projected costs for the project plan are compared with actual expenditure, using three values:

- budgeted cost of work scheduled (BCWS). This is the cost of tasks according to the project plan.
- budgeted cost of work performed (BCWP). This is the planned cost of work for tasks that have been performed.
- actual costs of work performed (ACWP). This is the actual cost of work for tasks that have been performed.

Two metrics are derived from these values:

- Cost difference = BCWP – ACWP

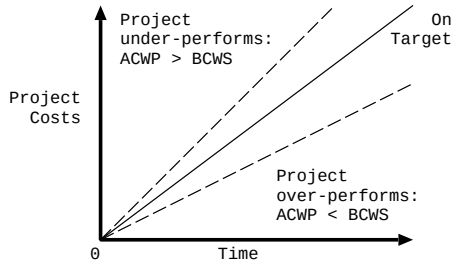


Figure 3.4: *Time and cost: How the financial measurements of BCWS, BCWP and ACWP indicate the financial progress of a project.*

- Schedule difference = BCWP – BCWS

Project progress is calculated by graphing the cumulative values for BCWS, BCWP and ACWP. At any point a project may be on schedule, ahead of schedule (the schedule difference is positive), or behind schedule (the schedule difference is negative), underspent (cost difference is positive) or overspent (cost difference is negative), see figure 3.4.

Project control is achieved by interpreting the differences. Lewis lists three situations:

1. ACWP is close to BCWP and larger than BCWS: extra resources have been assigned to the project, at budgeted rates.

2. ACWP is close to BCWP, and below BCWS: the project is either under–resourced, or slow to get started for other reasons. The INCIS project showed this behaviour during the first year.

3. ACWP is below BCWS and BCWS is below BCWP: the project is ahead of schedule and underspent. Labour estimates may have been padded, or the project might be unexpectedly easy to carry out.

Lewis ([49], p94) suggests acceptable differences for earned values of 3–5% for well–defined work such as construction, 10–15% for research and development, and "the sky is the limit" for pure research. For IT, where our

examples are drawn, Fenton [32] finds that estimating software costs is unreliable by factors of up to 200%, and Collins ([14], p23) lists a number of well–known software disasters and mentions that many of these could have been successes, *if the costs of the new systems had been calculated meticulously and then multiplied by three* (Collins' emphasis). These two large figures indicate the difficulty of estimating the costs of, and controlling, very complicated projects such as IT projects.

Keeping projects on track

Project management tools and techniques such as Gantt charts and earned value analysis are used to monitor deviation from the project plan. For example: the INCIS [79] project budget initially was underspent because of difficulties beginning the implementation, but later INCIS overspent its budget by a substantial amount (approximately three times the original estimates) as the project team desperately tried to get the software working.

Actual progress of a project may be on schedule, or run ahead of planned progress, or the project may progress more slowly than planned. The three ways of dealing with this situation are related to three facets of project management shown in figure 1.1 on page 2. More resources, if available, can be allocated to the task in an attempt to speed it up, although complicated projects such as IT projects may not be helped with such an approach [10][33][14]. Secondly, the estimated length of tasks can be changed, so that the schedule of the project is changed. The third alternative is to change the scope (the goals) of the project by deleting the task from the project and dropping the requirement that it satisfied. In such a scenario, we are faced with the requirement to change a project plan that, ideally, should not change. The project manager must make these decisions of how to revise the project.

Even if a risk management technique similar to NZS4360 [60] [1] is applied to the project plan, and tasks adjudged especially risky have risk mitigation or avoidance built in (eg: the work estimated for risky tasks is increased by some factor to account for anticipated slippage), there is still the likelihood that the actual project progress will deviate from the planned progress. Such

a risk–management approach also has a so–called opportunity cost[2]. Assisting the project manager revise a project is a research problem that will be further addressed in later chapters.

3.1.5 Closing a Project

A project might be closed in a number of ways. Ideally, a project is closed when the last task is 100% complete. However, a project might also be closed when either time or resources run out, or if the project progress has deviated too far from that planned. Projects terminated for lack of time or resources or poor progress are often failures, and their unachieved goals must be either abandoned or used to initiate another project.

Activities associated with successful project closure might be the hand over of the project results to an organization, for ongoing management. For example: an electrical appliance might be designed with a project team, but the design would be handed over to an appliance factory for production, after–sales support, etc.

3.2 Project Management Tools

Project Management software supports the creation, measurement and management of projects described above. For example: we have illustrated project concepts in figures 3.1 to 3.3 with screen shots from the freeware tool Planner [42]. Software tools such as Planner and Microsoft Project help a project manager deal with:

- Recording the tasks, with associated resource requirements and work estimates, etc, required to achieve the project goal.

- Recording the resources available to, or required for, the project and describing their availability with a calendar.

- Allocating resources to tasks, so that limited resources can be allocated in the most efficient way; normally the highest utilization. Allocations

[2] Opportunity cost is the cost of extra resources assigned to a project that turn out not to be needed.

may be made under the control of the Project Manager, or automatically according to an algorithm. Project tools can produce individualized schedules for each resource in the project, and flag times when resources are over-allocated (ie: used more than 100%) for the project manager.

- Scheduling of tasks and scheduling of allocations according to project and resource calendars.

- Providing template plans that can be filled in with task and resource information, and changed to suit the goals at hand. For example: templates are included with Microsoft Project for such projects as software development, construction and product introduction.

- Visualizing project history and status with a Gantt chart or other tool such as a PERT [66] chart.

- Summarizing project costs in various ways; according to a Work Breakdown Structure, for example.

- Summarizing actual progress against estimated project progress.

Software tools can usually present many different views of a project, such as a list of tasks, a list of resources, or a project visualization such as a Gantt or PERT chart. The user interface is typically some kind of direct–manipulation interface, so that the attributes of a task can be listed by double–clicking on a task. Since project management has existed in its modern form for decades, it is no surprise that the software tools associated with it have not changed fundamentally, and Gray and Larson ([39], p 379) wrote in 2000: "Differences among [project management] software in the last decade have centered on improving "friendliness" and output that is clear and easy to understand."

```
<?xml version="1.0"?>
<project name="New fridge design" company="Ace Appliances"
manager="F. Johnsen">
  <tasks>
    <task id="1" name="Brainstorming session"
    note="Initial ideas session" work="43200">
    </task>
    ...
  </tasks>
  <resource-groups/>
  <resources/>
  <allocations/>
</project>
```

Figure 3.5: *A sample of XML from a file produced by the Planner program representing a simple project. The project element, demarcated by tags, is the root XML element and has an attribute, "name", that has the value "New fridge design". The project element contains a task list in which one task is visible. Potential lists of resource groups, resources and allocations of tasks to resources are empty.*

3.3 Project Data Models

Many of the software tools discussed in section 3.2 can store project data in XML [93] for which there are published XML schemata [95]. Project Management tools generate XML for a variety of reasons: Planner [42] uses an XML format as its native storage for project data, as shown in figure 3.5. Microsoft Project [55] generates XML [58] for connectivity with the .NET architecture, and PMXML [68] is a vendor–independent XML schema for transporting Project data, developed by a consortium. We examined the XML schemata for a number of project software tools, looking for common features.

3.3.1 Microsoft Project Data Interchange (MSPDI) Schema

This XML schema was developed by Microsoft for Project 2002 and later versions, and is used to export XML data from the Microsoft Project software tool, normally saved in a proprietary format. Because of the popularity

of Microsoft Project, many other project management tools can import or export MSPDI data—Planner can import MSPDI data, for example. MSPDI is the largest of the project schemata that we studied, because it has many extra elements added to the basic project data model. For example: ten different baseline plans can be stored (a baseline is a snapshot of a project plan).

3.3.2 Project Management XML (PMXML)

A consortium headed by Pacific Edge Software Inc. and including NASA and Oracle created PMXML in 2000.

PMXML is a published standard, but only the X Schema Definition and brief documentation is publicly available; tools to use the standard are proprietary, such as the Project Office Connector for Microsoft Project 2002, a bi-directional PMXML–based interface for Project 2002. PMXML appears to be languishing as a standard[3], but it is an example of a data model for project data that attempts to be universal, and for this reason the Open Project Management Exchange Format project (OPMEF [2]) have considered the use of PMXML.

3.3.3 Planner

Planner uses XML for its native data file format, and the document type definition (DTD [28]) of this format is distributed with the (free) Planner software[4] [42]. The Planner data model is stripped down to the basics, consisting of little more than a project with tasks, resources and allocations of resources to tasks, and a project calendar.

---

[3] Rainer Volz [87] writes:

> PMXML is defined and built–in into some systems, at least at Pacific Edge Software and Primavera Systems, but there is not much information about its details, the goals and timeliness for its further development.

[4] The Planner DTD version 0.6 distributed with Planner version 0.13 doesn't describe all the XML data that the Planner software actually writes: the `Task` element definition in the DTD is missing the attributes `work-start` and `priority`, and the `Resource` definition is missing a `short-name` attribute.

Planner lacks many of the facilities of the more complicated software tools. For example: Microsoft Project and PMXML can process costs in a variety of foreign currency units. Planner uses no currency, only a numeric cost. Microsoft Project and PMXML can assign resources to tasks for limited periods of time, less than the total duration of the task. In Planner, this is not possible: allocations have no associated duration and so the allocation is for the entire length of the task. To achieve the effect of an allocation for a limited duration, it would be necessary to split the task into two or more sub–tasks of the desired allocation duration, and assign the resource to one or more of these. Planner has been maintained as an open software project for several years, and is distributed with the Fedora 6 Linux distribution, for example.

3.3.4 GanttProject

Ganttproject [84] is a Java application that has a similar appearance to other Project tools such as Planner or Microsoft Project. It is a very simple tool, lacking many of the extra facilities of more complicated project tools, but still containing the common features of tasks, resources and allocations, and because of its simplicity, Ganttproject is interesting to examine. Ganttproject reads and writes XML data, and the DTD is distributed with the software, which has been maintained for a number of years as an open–source project.

3.3.5 Comparing The Project Data Models

As shown in table 3.1, project software tools differ greatly in their complexity, and in the complexity of the data that they store. Our analysis of the facilities and XML schemata used for the tools Microsoft Project, Planner and GanttProject, and the public XML schema PMXML, shows that project tool data models essentially reflect the basic project management concepts described in this chapter: A project is made up of tasks, resources and allocations of resources to tasks.

We have modelled these essential concepts and their relation to each other in figure 3.6. The larger schemata have more elements and attributes than figure 3.6 because they offer more project reporting and analysis. For

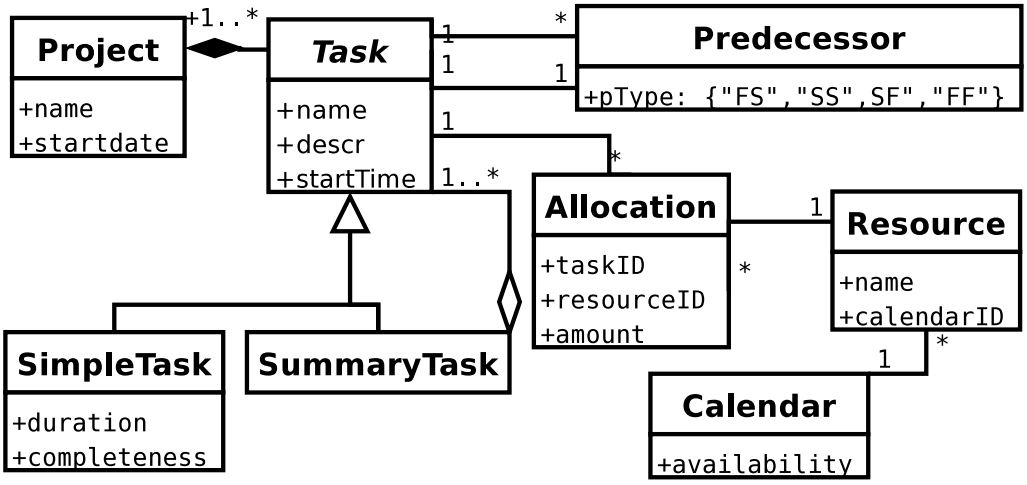


Figure 3.6: *Our conceptual UML [35] model of the data in the Project domain. Our model reflects the essential concepts of Projects and Project Management, found in all the example tools we studied.*

Schema Name	Schema size	Element defns	Attribute defns
Planner DTD	6 kB	27	72[5]
GanttProject DTD	1324 B	10	23
MSPDI 2002	151 kB	405	0
PMXML 1.0	44 kB	399	18

Table 3.1: *Relative sizes of the DTDs and XML schemata discussed. Planner makes use of XML attributes where the other schemata declare elements instead—Microsoft Project uses no attributes at all.*

example: MSPDI contains fields to store a customized Work Breakdown Structure, but Planner XML does not. PMXML allows many projects to be grouped together, but the other data models do not. Instead of the simple DTD used by Planner, Microsoft uses an XML Schema [95] defining a vast number of elements, where other schemata might use attributes associated with the elements. There is also a good deal of strong type definition in the MSPDI and PMXML schemata, something lacking in the Planner and GanttProject DTDs. Differences such as those above are reflected in the numbers of table 3.1, and figure 3.7 illustrates some of these differences.

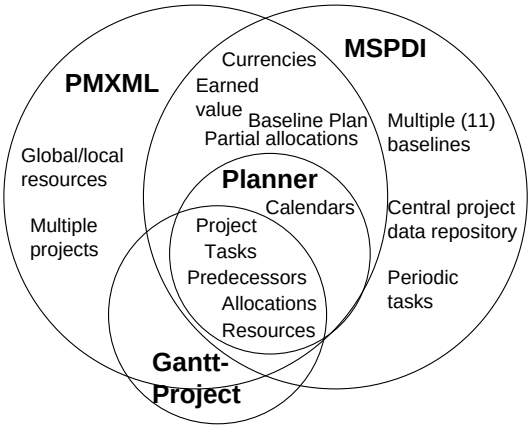


Figure 3.7: *A Venn diagram illustrating some of the differences between the data models of PMXML, MSPDI, GanttProject [84] and Planner. The Planner data model is a subset of both of the larger models because it represents nearly the smallest data model that can usefully describe project data.*

Implementing Project Data Models

Different project tools implement their project data model in different ways. For example: in section 3.1.2 we mentioned the idea of a summary task. Microsoft Project represents summary tasks by assigning each *Task* element a `WBS` (Work Breakdown Structure) element, the content of which assigns the *Task* a place in the WBS hierarchy[6]. Planner represents the `SummaryTask` relationship by nesting *Task* elements in an XML tree, so that the XML file structure reflects the nesting of `SummaryTasks`. PMXML uses a third way: each *Task* element has a `ParentTaskID` and a `IsSummary` element, so that the hierarchy of `SummaryTasks` is built up by storing the identifiers of parent tasks.

Extending Project Tool Data Models

The above schemata also include extra user–defined fields for user–specified data. An example of such usage might be to include a "sign off" attribute for

[5] Includes attributes left out of the DTD.

[6] The result of this implementation decision is that the Work Breakdown Structure defines the *Task* nesting for the project

each task: none of the data models discussed above include such an attribute, but the attribute could be included with a user–defined field. Even though the project tools would not support any semantics associated with the user–defined fields, external uses of the XML data are possible because XML is an open format and the structure of the data is described with XML schemata and DTDs. In our example, we could use an external tool to report tasks that were complete but not signed off. We will make use of this openness of XML data in later chapters, where we describe an implementation of our solution to the research problem.

Another example of using user–defined fields would be to simulate some of the functionality of the more complicated project tools in a simpler tool. For example, we mentioned that MSPDI contains fields to store a customized WBS, but Planner XML does not. However, we could add a customized WBS to Planner with a user–defined field.

3.4 Exchanging data between project tools

The data models examined above were created without reference to each other. However, because of the essential similarity between these data models, it is possible to exchange project data between the tools that use them. A simple tool may have to throw away data associated with the more advanced features of a complicated tool: Microsoft Project calculates and stores values for earned value analysis that Planner cannot represent, for example. The result is that information is lost, but provided that the simpler tool can use the essential concepts we have identified in figure 3.6, the resultant project data will still be useful for project management. A similar situation applies when a more complicated project tool imports project data from a simpler tool, but in this case many of the data fields in the more complicated model will have to be left empty, or with default values.

One way of implementing the transforms is with XSL transforms [96]. Any XML document can be transformed, both representationally and structurally, using an XSL transform that maps XML elements, attributes and structures from one form to another. In our case, transforming XML data from one project data model to another involves not only changes in the

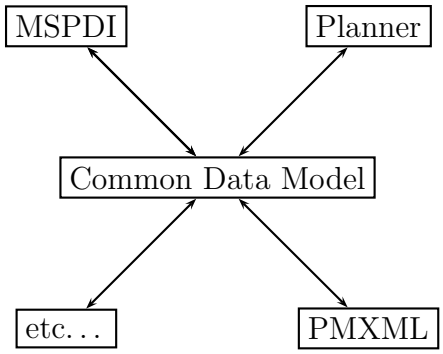


Figure 3.8: *Exchanging the essential concepts of project data (Tasks, Resources, Allocations, etc) between various PM tools, by transforming the data to and from a Common Project Data Model.*

XML document elements, attributes and structure, but also the conversion of, for example: date and time formats.

3.4.1 A Common Project Data Model Approach

One approach to exchanging project data is illustrated in figure 3.8. In this case, a central model is defined, and transforms to and from every other data model are defined. This method has the advantage that only *2*N* transforms are needed for N tools. Such efforts as the Open Project Management Exchange Format will use this approach of creating a universal project management data model to "create an open standard format for exchanging project management data". Similarly, and as we have seen, PMXML was created as a standard XML format for exchanging project data.

The results of this approach depend on how the "Common Project Data Model" is formulated: If the model reflects a common subset of all the data models (as our model of figure 3.6 does), then much data will be discarded at each exchange. If the model aggregates all the concepts in the tools then it will be huge, and will probably require augmenting for each new tool found—and this may require changing every other transform for every other tool.

For the purposes of generating project data the approach of using a minimal common data model has an important advantage: If we are able to generate project data (say: from a PMM) in terms of this model, then we

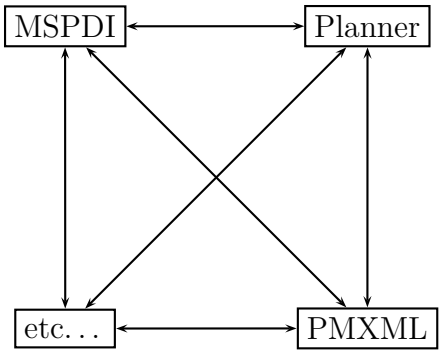


Figure 3.9: *Exchanging project data between various PM tools, by transforming from every data model to and from every other one. Each transform can be tailored to result in the least loss of information, but requires N*(N-1) transforms for N tools.*

will be able to generate project data usable by any project tool that can import the minimal common data model.

3.4.2 Experiments using a Model–To–Model Approach

Another way of exchanging project data is simply to transform the data directly from the model of one tool to that of another, as illustrated in figure 3.9. We used this approach to examine the transformation of project data between Microsoft Project, PMXML and Planner, as described in the following sections.

Transformation of MSPDI to Planner XML

Planner includes the facility to import MSPDI, and then save it in Planner's native XML data format. Little structural transformation is involved: the root element of both schemata is a `project`, for example. The great difference in complexity between the two schemata requires that the information associated with the vast amount of extra data stored by Microsoft project be discarded from the Microsoft Project XML file. This is in the order of 95% of the file for a typical project, but often much of this extra data has null or default values, and only the fundamental project concepts of tasks, predecessors, resources and allocations are set up by the user. For this rea-

son, it is still possible to produce a useful representation of a project stored in MSPDI format in Planner, with the above fundamental project concepts intact. One or two screen shots in this thesis, such as figure 4.2 on page 38, were generated from project plans transformed in this way.

Transformation of Planner XML to MSPDI

The Microsoft Project schema is vastly more complicated than that of Planner, so that even the file for an empty project is over 7 kB in size, containing over 200, mostly empty, elements. The majority of the extra information does not exist in Planner, and so it must be created in the form of elements which are either empty or which contain default values. We wrote an XSL transform to map Planner projects to MSPDI, along these lines. Projects created and saved from Planner could be then transformed to the MSPDI format, and read into Microsoft Project.

Transformation of PMXML to Planner XML

We did not have direct access to any PMXML–capable project tools. However, the PMXML web site made available the PMXML schema and sample project data files. We wrote an XSL transform to transform the PMXML project data to Planner XML. As for MSPDI, a large amount of project data had to be discarded. However, a recognizable project could be read in to Planner from the transformed PMXML project data. PMXML has the odd feature of not using predecessors directly, instead there is a `TaskOrderID` attribute to order tasks. However, it is possible to order tasks using the start and finish times stored with each task in PMXML, and so these tasks were transformed to Planner tasks set to start and finish at the specified times.

3.5 Conclusion

In this chapter we have briefly described project management (PM), how PM works, and how it is supported by project management tools. We have also described how projects are structured and measured, and how such tools as Gantt charts provide history and context for a Project. The project tools

we examined use a fundamentally similar data model of tasks, resources and allocations that we described, and will use again in later chapters. This similarity allows the exchange of the most important project information between project tools using, for example: XSLT. In the next chapter, we will examine how projects are created and changed.

Chapter IV

Project Management in Context

In this chapter we describe how project plans are formulated and changed. To do this, project management is informed by the knowledge of the particular domain the project is working in. We use a simple example to illustrate the situation when a project instance is produced and changed by hand, by filling in a template, or by following a Project Management Method (PMM)—a kind of "project recipe" that attempts to formalize the specialist knowledge used for a project.

4.1 Specialist Project Knowledge

Any project works in the context of some domain of knowledge, such as building construction, software engineering, etc. A project requires specialist knowledge to create the project plan (this knowledge could be encapsulated in a template plan, for example, as we describe in section 4.1.2) and then to manage the project once it is underway.

This specialist project knowledge is obtained by augmenting the judge-

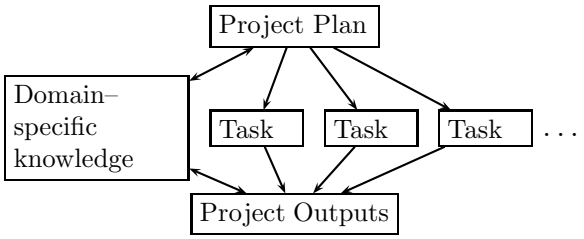


Figure 4.1: *Applying specialist knowledge to a project.*

ment of the project team with the knowledge of individuals skilled in the area in which the project is undertaken, and this is conceptually shown in figure 4.1. An example of the application of specialist knowledge is that estimating the work required for a particular task requires domain–specific knowledge of the kind of work involved in the task.

We have seen in chapter 3 that a project plan consists of a list of tasks, etc, that has the aim of achieving a defined goal in a limited time frame. In the following sections we list the methods of applying specialist knowledge to projects to create and/or change a project plan.

To illustrate the various ways of applying specialist knowledge to create a project plan, we will use a simple example. A software project was chosen for the example because not only do project management techniques apply to it, but also specialist software engineering PMMs such as the Waterfall Method can be applied.

The initial goal of the project as requested by the customer is to create a "hello world" program. The program then displays "goodbye world" when a button is pressed. The customer then adds an unforeseen complication to this simple goal: when they see the program working, they ask for the functionality of the program to appear in a browser window, rather than as output from a command–line.

4.1.1 Hand–crafted Projects

One way of applying specialist knowledge to projects is to manually apply the skill and experience of the project team. The team follows the procedure illustrated in figure 1.2 on page 2 to create a project plan, and then to manage the project to a conclusion. Software tools can assist the project team, but something these tools cannot do is encapsulate the specialist knowledge needed for creating and managing a project. If we were to manually apply conventional PM to the production of the "Hello World" program then we might formulate a project plan similar to this:

Project start: May 11, 2004, 10am.

Task 1: Initiate the "Hello World" project. *Requires a Customer*

Task 2: Specify the "Hello World" program. *Requires a Customer and a Programmer*

Task 3: Implement the "Hello World" program. *Requires a Programmer and a computer*

Task 4: Test the "Hello World" program. *Requires a Customer, a Programmer and a computer*

In the case of our example, we are in a quandary when the acceptance test of task 4 fails: the plan does not admit this possibility. Perhaps a "Change the program as needed" task could have been included in the original plan after task 4, but that might not be enough—a complete re–implementation might be required, for instance. Extra tasks, resources and allocations may be inserted into a real project, and so there is deviation from the plan. Conventional PM provides tools to report on the amount of variance of real progress from the project plan, as we described in chapter 3, but the project manager must decide for themselves how to change the project plan.

Software tools for conventional PM can automate the production of project reports, and make the revision of plans less laborious. However, they do not assist with decisions about how to change the plan.

In the case of our example, we might decide to simply restart the project. A restarted project might benefit from the initial attempt, but in our simple example the first project is a dead loss. If the second acceptance test passes, we would end up with a project plan as follows:

Project start: May 11, 2004, 10am.

Task 1: Initiate the "Hello World" project. *Requires a Customer*

Task 2: Specify the "Hello World" program. *Requires a Customer and a Programmer*

Task 3: Implement the "Hello World" program. *Requires a Programmer and a computer*

Task 4: Test the "Hello World" program. *Requires a Customer, a Programmer and a computer*

Task 5: Restart the "Hello World" project. *Requires a Customer*

Task 6: Specify the "Hello World" web page. *Requires a Customer and a Programmer*

Task 7: Implement the "Hello World" web page. *Requires a Programmer and a computer*

Task 8: Test the "Hello World" web page. *Requires a Customer, a Programmer and a computer*

The last four tasks were not planned for, but were required to be added to the project in order to achieve a successful result.

4.1.2 Template Project Plans

Although project tools cannot automatically create a project plan to achieve a specified goal, they do allow the use of template plans so that a project manager can quickly and conveniently create a project plan. A template is a project plan that requires a project manager to "fill in the blanks", with estimates of the work required for tasks and lists of actual resources. The template will normally be augmented with some textual description to indicate how to use it, and what goal it is for: it may be titled "Template plan for moving house", for instance. Resources could have some indication of their function added as a name ("Seller's Lawyer") or description ("The seller's conveyancing lawyer"). Because project tools lack the ability to store specialist project knowledge, it's necessary to add these descriptions to indicate how to use the template.

Microsoft provides a generic software project plan template with Microsoft Project, illustrated in figure 4.2. The template consists of a number of specified tasks and resources that have names and/or descriptions that look like they belong to a software project. The project manager can use this template quite well if they have some knowledge of software projects. They

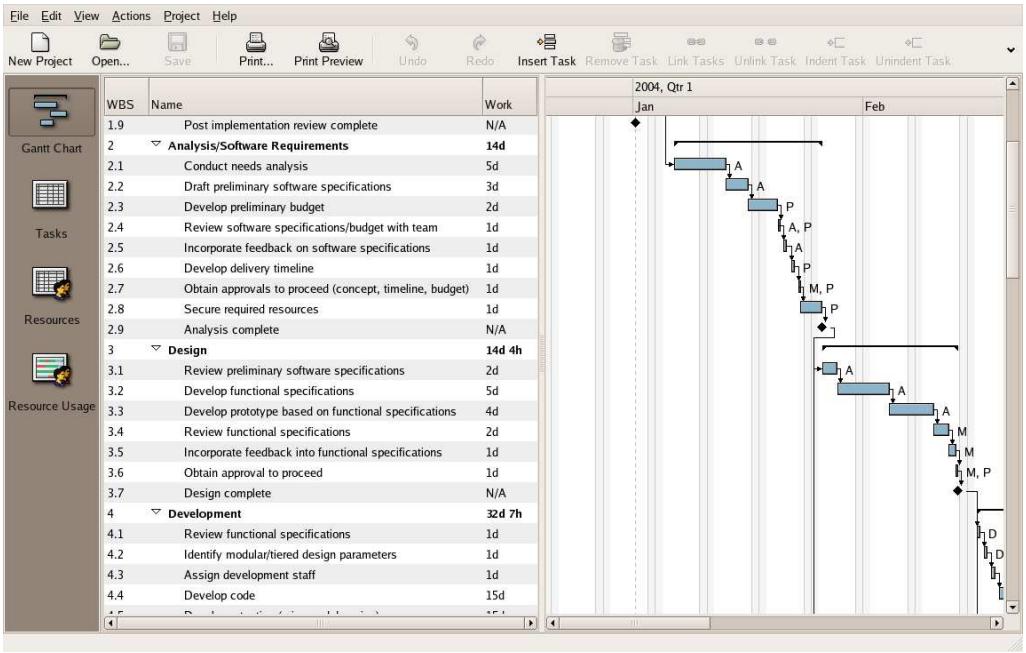


Figure 4.2: *The Microsoft Project software development template, exported as MSPDI XML and imported into Planner for display.*

must have this knowledge because some skilled interpretation is required to fill in the template. Documentary descriptions might be available with the template to advise the project manager how best to use it.

For our "Hello World" example, filling in the template results in a project plan that looks like a software project. However, we have no clue about how appropriate our use of the template really is. Are all the tasks in the template needed for our simple goal, for instance? Human interpretation is required to decide this. The template also provides no specific advice about what to do if the actual progress begins to deviate from the plan: should we restart the project, or insert tasks into the project to accommodate the changes? These are not faults with Microsoft Project, or the template. The PM domain, in which this software tool works, works best when a project proceeds according to a predefined project plan. Although templates can be useful for creating these project plans, there is still a reliance on the project team's skill to create a workable project plan and successfully manage the project.

4.1.3 Project Management Methods (PMMs)

Project templates contain textual descriptions of the kinds of tasks and resources required for a particular goal, and this suggests a strategy for describing specialist project knowledge: If we take these textual descriptions, then we can create a kind of "project recipe", that tells us about the kinds of tasks and resources needed to achieve a specified goal. We will call this "recipe" a Project Management Method, or PMM.

A simple way to describe PMMs is to write them down in English as a series of steps. Steps are not project tasks: they are a *specification* for a task, and can be used again and again, as opposed to a one-use task. For our example, we will use a simple method described by Schach ([76], pp 64–5) to write a small software program according to a customer's specification:

> The name of the method is "Build and Fix".
>
> 1. The Customer and Programmer initiate the project, and then
> 2. Iterate the following sequence of tasks until the Customer accepts the program in step 2c, or they halt the project:
>   (a) The Customer specifies the program to the Programmer.
>   (b) The Programmer implements the program.
>   (c) The Programmer presents the program to the Customer for acceptance.

Following the above description results in tasks with resources allocated, in a similar way to a project plan, but there is a crucial difference between applying a PMM and creating a project plan or using a template plan: the PMM reacts to the state of the project to control the production of tasks, depending on the outcome of particular steps on the PMM.

In our simple example, the Build and Fix method cycles through the main sequence of steps and results in the production of a "Hello World" program. At the acceptance test, the customer rejects the program, but decides to continue the project. The Build and Fix method then requires a re specification of the project (as a "Hello World" web page) and a second

implementation and acceptance test, which now passes. The resultant project consists of the following sequence of tasks:

Project start: May 11, 2004, 10am.

Task 1: Initiate the "Hello World" project. *Requires a Customer*

Task 2: Specify the "Hello World" program. *Requires a Customer and a Programmer*

Task 3: Implement the "Hello World" program. *Requires a Programmer and a computer*

Task 4: Test the "Hello World" program. *Requires a Customer, a Programmer and a computer*

Task 5: Specify the "Hello World" web page. *Requires a Customer and a Programmer*

Task 6: Implement the "Hello World" web page. *Requires a Programmer and a computer*

Task 7: Test the "Hello World" web page. *Requires a Customer, a Programmer and a computer*

Following the PMM has resulted in a series of tasks that achieve the desired goal. The resultant project tasks are quite similar to those of our hand–crafted project plan of section 4.1.1. The crucial difference is that the PMM specified what tasks were to be added, so that the project was revised in a controlled way. We could say what tasks had to be added (tasks 5–7), and why (because the acceptance test failed, and the customer decided to continue the project). This is a capability that project plans, even template plans, do not have.

One objection to the above approach is that the use of a PMM to drive the project results in a plan that always changes, although the changes are always constrained. Iterative PMMs such as XP [5] can work around this

limitation, by guaranteeing to deliver a working system no matter when the project is halted (eg: by budget limitations). However, in this case the project scope is not fixed in advance.

4.2 Observations

Our simple example has illustrated the requirement that any real project has for specialized knowledge. We have tried three ways of applying this knowledge to a project, and shown that two of them: hand–crafting a project plan and using a template, rely to a large extent on the skills and experience of the project team and/or the template designer to create a workable plan. When we were faced with changing requirements from the customer, both the hand–crafted and template–driven project plans required ad–hoc revisions to produce a successful result.

The third way of creating a project plan, using a PMM, was different to the first two, because the PMM specified how to create the project instance. When the customer's requirements changed, the PMM specified how to cope with this situation. A PMM is not a guarantee of project success, but in situations where the project has to change (eg: if the requirements are changed), a PMM can assist a project manager to revise the project.

4.3 Conclusion

In this chapter we have introduced the concept of applying specialist knowledge to projects, and described three different ways of applying this knowledge: manually, using a project template, and by using a Project Management Method (PMM). We have then briefly described PMMs, and in chapter 5 we will analyze PMMs more closely, using the example project described in this chapter.

Chapter V

Constructing a PMM Vocabulary

In this chapter we use the example project of chapter 4 to analyze the application of three Project Management Methods (PMMs) of increasing complexity. The resulting project instances are illustrated using a simple two–dimensional visualization that shows the PMM steps executed against time. These examples are then used to draw some general conclusions about the PMMs, that will be used to create a PMM data model.

5.1 A Simple Visualization of the "Build and Fix" Method

In section 4.1.3 we produced a project instance using Schach's "Build and Fix" method. The resultant project is visualized in figure 5.1, with the project tasks represented in a graph. The vertical dimension is the names of the steps used by the PMM and associated with each task, and the horizontal dimension is time, showing the chronological sequence of tasks. Required

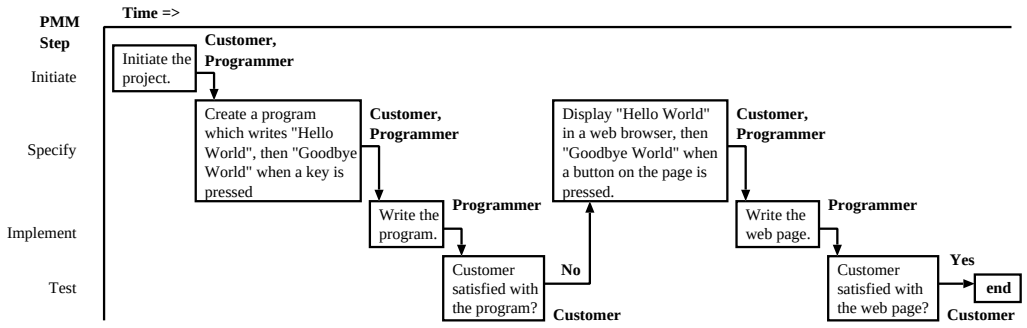


Figure 5.1: *The history of tasks for the project produced by applying the "Build and Fix" method to the simple example.*

resources, such as programmers, are listed against their associated steps. This visualization is similar to a Gantt chart, that plots tasks against time, but in this case we are plotting PMM steps against time, so that in figure 5.1 two diagonals occur, when the PMM loops around and repeats the same set of steps. Because it lists only tasks, a Gantt chart would interpret each task as unique, and therefore would show this sequence of tasks as one diagonal (as shown later in figure 5.5). Our visualization is able to show how project tasks relate to PMM steps, and we will make use of this property to formulate some observations about PMMs.

Recognizing features The successful completion of the project results in the production of a working web page which satisfies the customer requirements. This is a different goal to that originally specified—writing a "Hello World" program—and the project steps had to be repeated to accommodate this. In figure 5.1 the PMM iterates twice through the sequence of project steps, terminating when the customer is satisfied.

The application of the sequence of PMM steps produces a characteristic diagonal pattern of tasks. Repeating the steps results in the sequence of tasks beginning again from the first (topmost) step. Our visualization thus shows two distinctive control structures:

Sequence The Build and Fix steps follow each other in a sequence, and so do the resultant tasks.

Iteration At the acceptance test, the decision is made whether to continue to the next step, or go back and repeat the PMM from this or an earlier step.

In section 5.5 we will describe more control structures, but first we will apply two more realistic PMMs to our simple example: the Waterfall Method and Extreme Programming.

5.2 Applying the Waterfall Method

We now apply Royce's waterfall method [73] to the example project. This method uses a sequence of six steps ([76], pp 65–70), and we describe it in

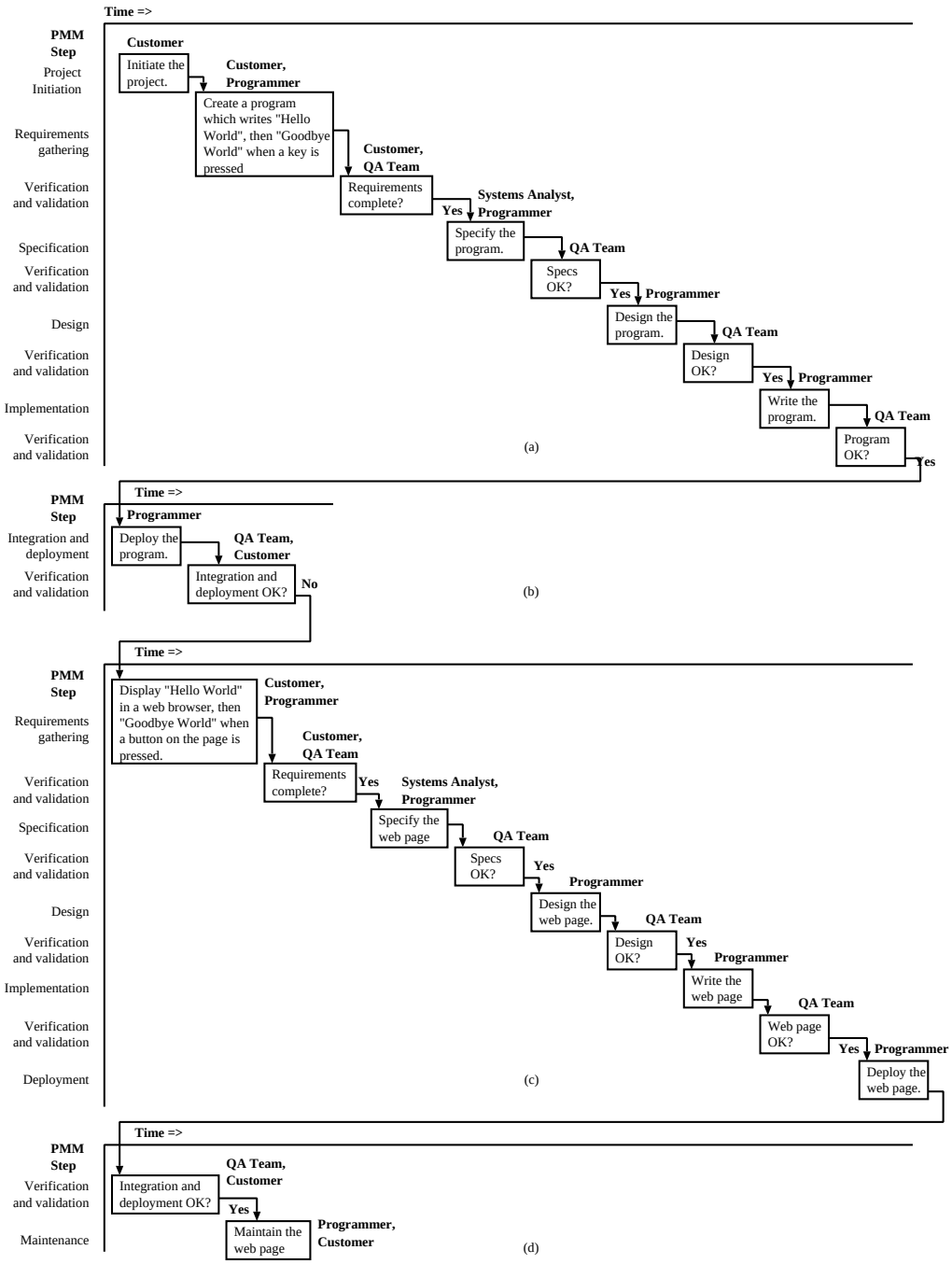


Figure 5.2: *Visualizing the application of the Waterfall method to produce project tasks. In our example the method restarts in part (c), when the initial implementation does not pass the customer's acceptance test.*

appendix A. The waterfall method has well–known disadvantages ([24], Ch 4), for example: no implementation is produced until close to the end of the project.

The Waterfall method has extra specification and design steps that are meant to ensure that the customer, and programmers, understand exactly what the implementation is going to produce. However, for this example, let us assume that the customer is happy with the initial specification and design stages of the project, but that when they see the program working they request the program be changed to a web page. Hence, two sequences of tasks are generated when the PMM iterates twice through, as shown in figure 5.2.

When the Waterfall method, with its long sequences of steps, is applied to the example project, many more tasks are produced than the simpler Build and Fix method. The extra design and specification steps should have ensured that the method didn't need to iterate again, but since they did not the impact of the initial rejection of the "Hello World" program is higher. More work has to be re–done, and this larger project results in the accomplishment of the same final goal, the production of a "Hello World" web page.

5.3 Extreme Programming

Let us now apply the Extreme Programming method [5] to the project. This method is described in appendix A. The list of tasks shown from the application of Extreme Programming is shown in figure 5.3. In our example, this characteristically iterative method produces many loops, with their associated tests.

The only released software that results from the application of XP is a web page—there is no stand alone program as created by the first two methods. This is because of the XP practice of breaking up projects into small parts ("user stories") that can be implemented independently. When the customer sees the implementation of the first feature and says that it is wrong (in figure 5.3 (b)), the project is halted and redesigned at that point, not after the software is released. Such early customer feedback is a valuable feature of XP, which has the recommendation "release early and

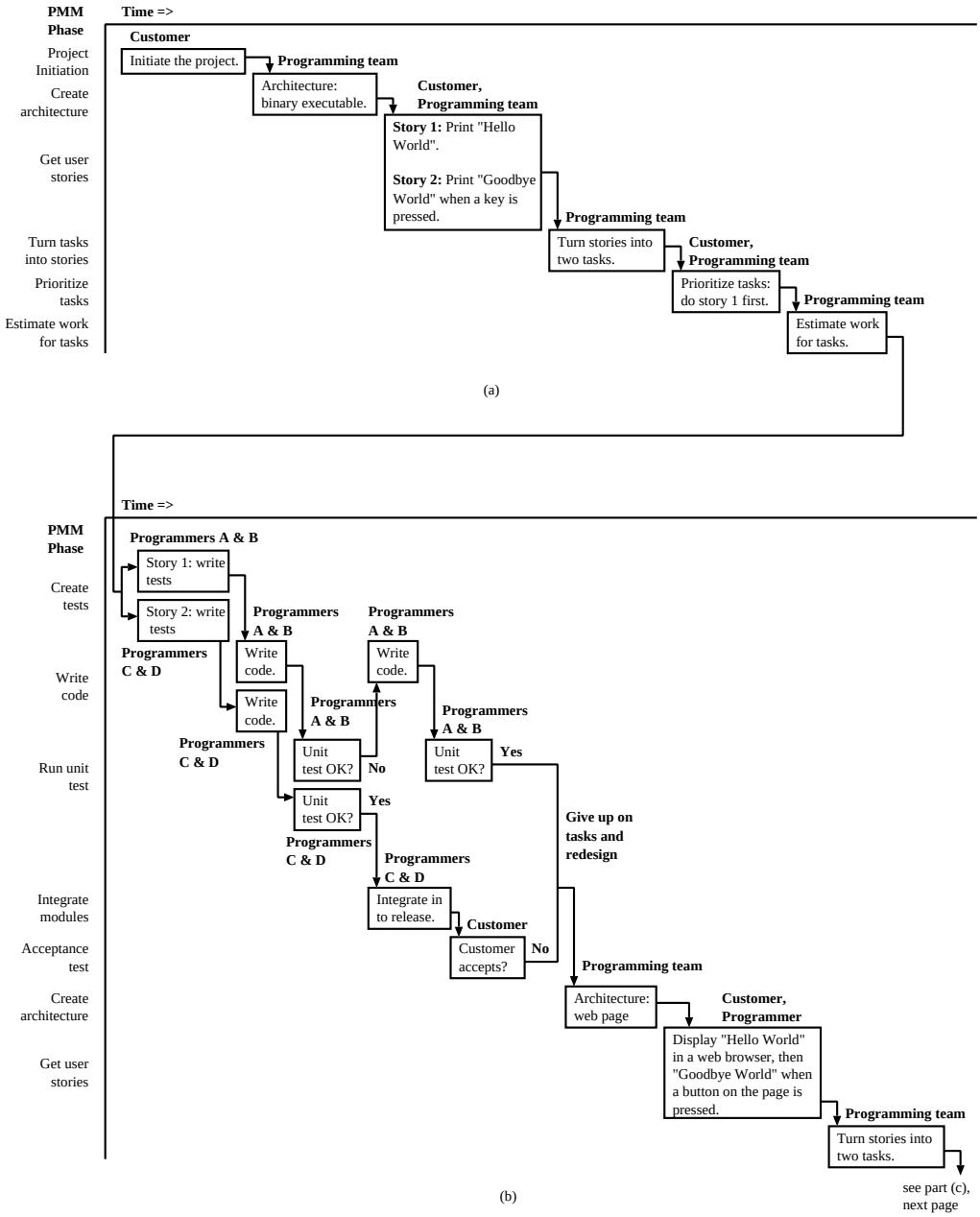


Figure 5.3: *Applying the Extreme Programming method to produce a project instance.*

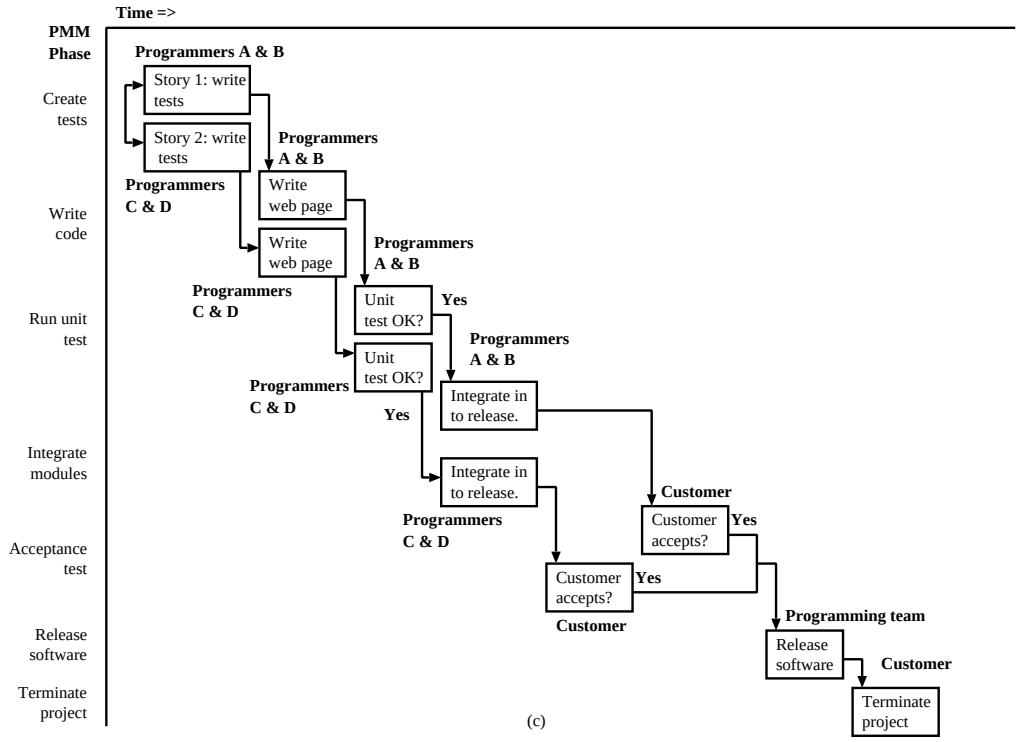


Figure 5.4: *Applying the Extreme Programming method to produce a project instance, cont'd*

release often".

Another result of the creation of user stories is that it allows the parallel application of tasks during the test–driven implementation step, where the implementation of stories one and two are undertaken simultaneously.

5.4 Observations

In sections 5.1, 5.2 and 5.3, we have described three PMMs and used a simple visualization to show the project instances they produce when applied to a simple example. The visualizations illustrate the following:

Different PMMs produce different project instances All the PMMs produce different project instances for the same goal. The build–and–fix method produces a small and simple project instance in section 5.1 that ac-

Resource Types Identified

PMM	Customer	Programmer(s)	Systems Analyst	QA Team
Build and Fix	*	*		
Waterfall	*	*	*	*
Extreme Programming	*	needs 2+		

Table 5.1: *Resource requirements of the various PMMs in the simple example project.*

complishes the simple project with the fewest tasks. The larger and more complicated Waterfall and XP PMMs produce larger project instances. However, for large projects the build–and–fix method lacks the formal control mechanisms, such as test–driven programming, of the more complicated PMMs.

Different PMMs have different patterns of resource usage Project tasks require resources of particular types, and this is illustrated in table 5.1, showing resource types identified by the three PMMs in the example.

The table shows that the resource requirements depend on the type of PMM chosen to run the project. Certain PMMs require particular resources or personnel allocated to particular tasks, for instance: the Waterfall method requires software quality assurance personnel to validate each major phase in the project, whereas the simple Build and Fix method requires only a customer and programmer.

5.5 PMM Control Structures

We have described several PMMs and then applied them to a simple example project. By applying a simple two dimensional visualization of the resultant tasks, we can discern the following features of PMM control flow:

PMMs use sequences of steps The "specify, implement, test" sequence of the Build and Fix method is a sequence of steps, which relates to the

generic PM concept of **precedence**, in that certain steps must be performed one after the other. A sequence of tasks may also be an artifact of iteration, for example: iterating the "code and test" loop of the XP PMM produces a sequence of tasks, although the associated control structure was not a sequence of steps.

PMMs use selection In figure 5.3 (a), a decision is made whether to have a "design spike" or not. The result of the decision is negative, and so the only step we see in the figure is the decision, not the "design spike" , which was not selected to occur. PMMs require decisions to be made at certain points, and these decisions control which steps or tasks are produced by the PMM and which appear, or do not appear, in the project instance.

PMMs use repetition The first PMM examined in the example consisted of a simple list of steps which produced a sequence of tasks. However, the Extreme Programming PMM is intrinsically iterative and makes use of a regular cycle of software releases to inform the customer about the current state of the project, by deploying the software. This leads to the concept of an **iterator** which unrolls a loop in the PMM, producing one or more sequences of tasks over time.

PMMs Use Concurrency The unsequenced tasks of the XP method shown in figure 5.3 (c) have a control flow which forks and then synchronizes two sequences of tasks, at the software release step.

PMMs nest control structures The characteristically iterative Extreme Programming PMM produces not only loops, but also loops within loops. Figure 5.3(b) shows the code for "Story 1" failing its unit test, with a resultant re–writing of the code and a re–running of the test. This minor loop happens within the main release loop of figure 5.3(b).

5.5.1 Comparing PMMs to PM

Figure 5.5 illustrates the basic PMM control structures in our simple visualization, and compares this to the Gantt chart representation used in PM. The

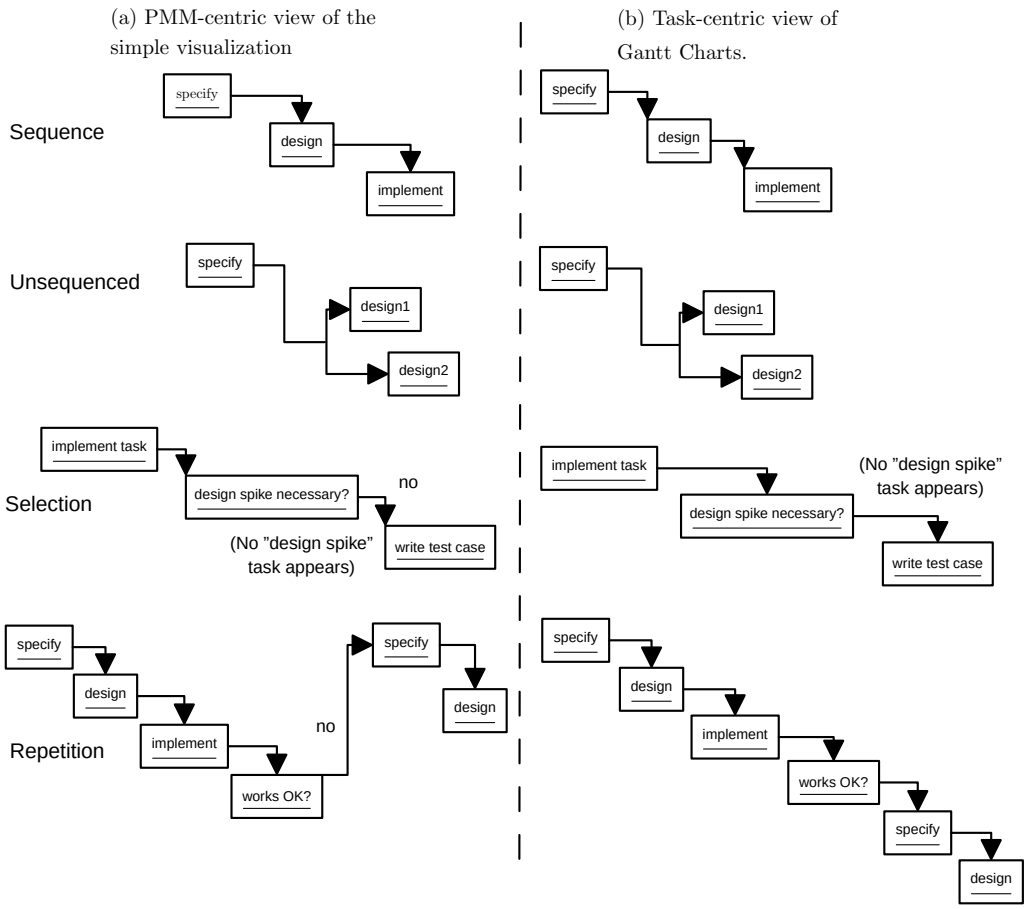


Figure 5.5: *Comparing Gantt charts with the simple PMM visualization to show the fundamental control structures of PMMs.*

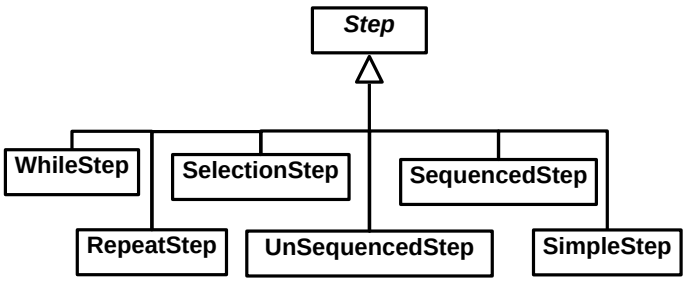


Figure 5.6: *UML defining as PMM Steps the different kinds of control structures we identified.*

difference is especially noticeable for repetition, which re–visits the same step twice in the visualization, but produces what looks like one long sequence in the Gantt chart. The application of the selection construct may require that some PMM steps (of the path not selected for) are never performed, and so do not appear in the visualization. For reasons such as those above, PMM control structures cannot be represented by PM software tools such as Planner, even though these tools can still represent the tasks resulting from the application of a PMM.

5.6 Formulating a PMM meta model

To construct a model that can express PMMs, we will use the unified modelling language (UML), and begin by modelling the various control structures we identified in the previous sections. These control structures reflect the programming constructs identified by Boehm and Jacopini [6]: sequence, selection and repetition. The authors found that computer programs could be represented by an arbitrary nesting of these control structures. With the addition of concurrency [77], we can similarly represent PMM control flows with an arbitrary nesting of sequence, selection, repetition and concurrency. Figure 5.6 illustrates the PMM step types we define to describe the following control structures:

`SequencedStep` for a step that applies a sequence of steps.

`UnSequencedStep` for a step that applies steps concurrently.

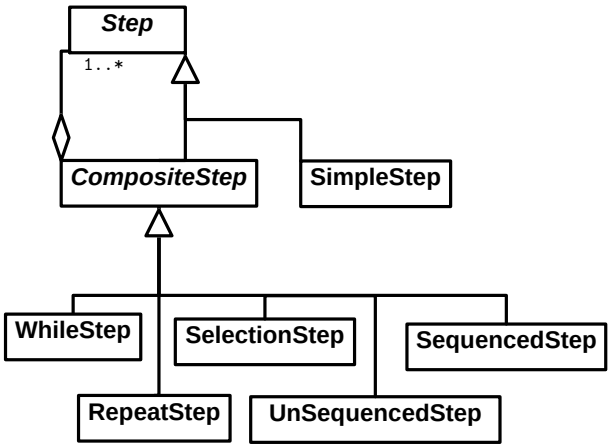


Figure 5.7: *Applying the Composite design pattern to allow the arbitrary nesting of PMM control structures.*

`SelectionStep` for a step that applies a step selected from a set of alternatives.

`WhileStep` **and** `RepeatStep` for a step that applies steps repetitively. The `WhileStep` applies a test before applying any steps, the `RepeatStep` applies the test after applying the steps.

`SimpleStep` for a step that specifies some kind of project activity.

To allow the arbitrary nesting of PMM control structures, we now apply the Composite design pattern ([36] pages 163–173) to our PMM step types. This pattern provides a tree–structured decomposition of an arbitrary nesting of different kinds of objects—in our case, PMM steps. Applying the Composite pattern to PMM steps produces the UML of figure 5.7. The abstract class *CompositeStep* is used to indicate that the abstract class *Step* may be a `SimpleStep` or an arbitrary nesting of other *Steps*.

`SimpleSteps` specify activities that require various kinds of resources to complete. We will define a `ResourceType` class that does not list literal resources, but instead defines what kind of resource is needed by a `SimpleStep`. Many different `SimpleSteps` might need many different `ResourceTypes`, ie: this is a many–to–many relationship, and so we must create an association class; `Requirement`, to reflect this relationship, shown in figure 5.8.

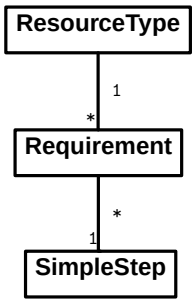


Figure 5.8: *Linking SimpleSteps to ResourceTypes with a Requirement class to reflect the many–to–many relationship.*

Finally, a PMM begins by applying a main *Step*. This relationship is shown in figure 5.9, that also combines all the other features of a PMM we identified above.

5.7 PMM Behaviour

The UML of figure 5.9 describes PMM concepts, but it does not describe the behaviour of a PMM when it is applied to a goal. Applying a PMM to a goal results in the creation of a project that contains tasks, resources and allocations. We can briefly describe how this happens, as follows:

Apply the PMM. Applying a particular PMM to a chosen goal results in the creation of a project that will list the resultant tasks.

Apply PMM Steps. Beginning with the first PMM step, the control flow of the PMM steps constrains the production of project tasks. `SimpleSteps` produce a single project task, `SequencedSteps` produce sequences of project tasks, and so on.

Specify Resources. The completion of a project task requires that one or more resources be used. PMM `ResourceTypes` specify the different resource types needed for each task resulting from the application of a `SimpleStep`.

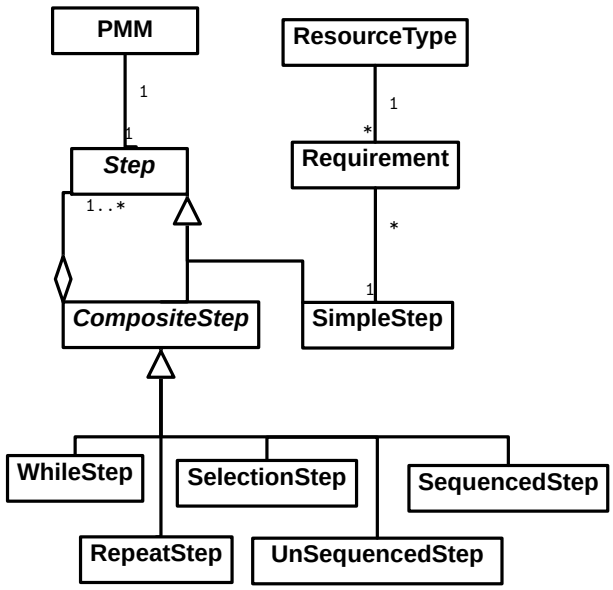


Figure 5.9: *UML diagram of PMM concepts, formulated from our observations of PMM behaviour.*

To specify PMM behaviour in more detail, we will first create a detailed description of the connection between our PMM and PM models. This will be done in chapter 6.

5.8 Conclusion

In this chapter we have used a simple example project to illustrate the operation of three Software Engineering PMMs. Using a simple visualization, we can find concepts of sequence, selection, repetition and concurrency in PMMs, and have used these observations to formulate a conceptual model of PMMs. In the next chapter, we will link this model to our PM model of chapter 3, to create a framework that links projects and project management methods.

Chapter VI

The PM/PMM Framework

In this chapter we will connect the PM and PMM models previously described to create a linked framework. We propose this framework as a solution to the research problem posed in chapter 2: the creation of a database–level connection between PM and PMMs.

6.1 Summary of the PM and PMM models

In chapters 1 and 3 we showed how PM, as supported by common tools, uses a one–shot approach to achieve a desired goal. A static project plan is formulated that describes a project that has tasks with allocations of resources. Project progress is monitored against this static plan, with the aim being to minimize the deviation of actual project progress from the plan. We then showed how the data models of some common PM tools were similar enough that we could, for example: usefully exchange project data between Microsoft Project and Planner, and created a meta–model of the essential common features of these data models (figure 3.6 on page 27).

In chapter 5 we showed that a PMM is a kind of "project recipe" that can be used repeatedly, as opposed to a one–shot Project plan, and that applying a PMM to achieve a desired goal creates tasks, resources and allocations, that can be considered as a Project instance. We then described PMM control flow in terms of an arbitrary nesting of sequence, selection, repetition and concurrency, and applied the Composite design pattern to produce a meta–model to represent PMMs (figure 5.9 on the preceding page).

We described PMM behaviour as the production of a project with tasks, resources and allocations. Because one PMM can be applied repeatedly, there is a one–to–many mapping from the PMM domain to the PM domain.

We can describe this mapping between PMMs and PM in more detail by considering that:

- one PMM may map to many Projects
- one PMM *Step* may map to many Project *Tasks*
- one PMM `ResourceType` may map to many Project `Resources`
- one PMM `Requirement` may map to many Project `Allocations`.

In the next sections, we will create a detailed model of these mappings, and use them to link the PMM and PM models in a framework.

6.2 A Framework that maps PMM to PM

Our requirements for the framework we propose are as follows:

Firstly, the research problem posed in chapter 2 is to link PM and PMM data so that, for example, projects can be described by a PMM. To address this problem, we will create a framework that maps our PMM model of chapter 5 to our PM model of chapter 3—the "database level" linkage described by Bussler [12].

Secondly, we wish the framework to be general purpose, so that potentially any kind of PMM and PM tool can be linked together, provided that the data can be described by our framework[1]. Consequently, PMM tools created using our framework will be generalized, not only in the PMMs they can describe, but also in the PM tools they can exchange data with. To be general purpose, we also require our framework to be independent of any particular architecture, such as J2EE, .NET, etc.

[1] Such a requirement is known to Software Engineering as separation of concerns: In our case we wish to keep PMM concerns on the PMM side of the framework, and PM concerns on the PM side of the framework.

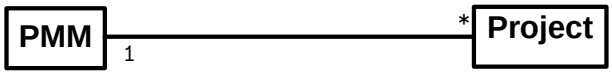


Figure 6.1: *Linking PMMs (left) to Projects (right).*

Thirdly, we wish to make the framework extensible, so that new concepts can be added as necessary.

In section 6.1 we identified four relationships that map PMM data to PM data. We will now consider each of these mappings in more detail.

PMMs ⇒ Projects

When a `PMM` is applied to a goal, a `Project` instance is created. This happens every time that a PMM is applied. We show the PMM–to–project link in figure 6.1.

PMM Steps ⇒ Project Tasks

We must be able to record any kind of *Step* to *Task* mapping that may occur in our framework. *Steps* may contain other *Steps*, and to reflect this nesting we will make use of a common device in the PM tools we studied, the `SummaryTask`. `SummaryTasks` provide a handy way of hiding details of project tasks, by "rolling up" a series of `SimpleTasks` into one `SummaryTask`. We will take the mapping of a PMM *Step* to many Project *Tasks* described previously, and subdivide it as follows:

- one PMM `SimpleStep` may map to many Project `SimpleTasks`
- one PMM *CompositeStep* may map to many Project `SummaryTasks`.

This use of project `SummaryTasks` allows us to map PMM *Steps* composed of *Steps* to project *Tasks* composed of *Tasks*, and so reflects the nesting of PMM *Steps* in a corresponding nesting of Project *Tasks* This relationship is shown in figure 6.2.

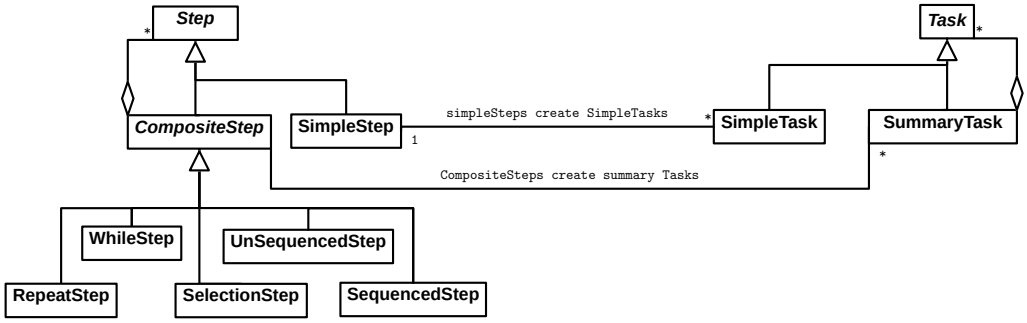


Figure 6.2: *Linking PMM Steps (left) to Project Tasks (right).*

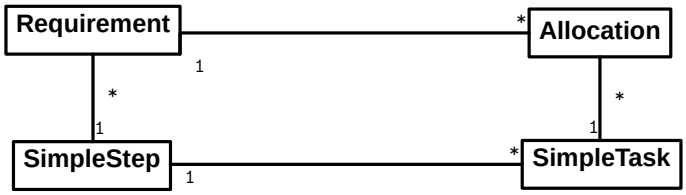


Figure 6.3: *Linking PMM Requirements (left) to Project Allocations (right).*

PMM Requirements ⇒ Project Allocations

Any `SimpleStep` may have associated `Requirements` for `ResourceTypes` that we will map to `Allocations` of Project `Resources`, as shown in figure 6.3.

We have chosen to link `SimpleSteps` to `Requirements`. Another option would have been to link `Steps` to `Requirements`, but this would have caused semantic ambiguity: what would such a construct mean for any steps nested within a `CompositeStep`? Should they inherit the `Requirements` of their parent, for example? In practice, we found that we could implement our choice without loss of generality of the data model.

PMM ResourceTypes ⇒ Project Resources

We classify Project `Resources` as having a particular `ResourceType`, which is used to specify what kind of `Resource` is given an `Allocation` to a *Task*. This relationship is shown in figure 6.4.

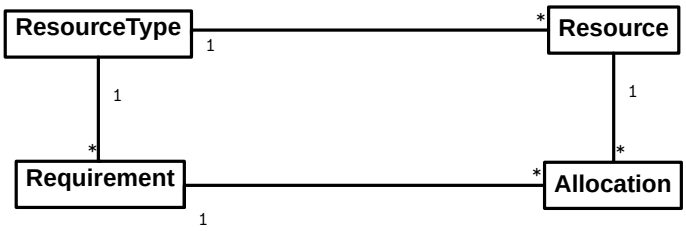


Figure 6.4: *Linking PMM ResourceTypes (left) to Project Resources (right).*

6.2.1 Constructing the Framework

Having described the mappings from the PMM domain to the PM domain, we can now model a framework that links the PMM and PM domains at the database level of Bussler's taxonomy, as we proposed in chapter 2. This framework model is shown in figure 6.5.

Our framework reflects the PMM–to–PM relationship in its objects, so that, for example: there is a one–to–many mapping from PMM `Requirements` to project `Allocations`, and PMM `ResourceTypes` to project `Resources`. A more subtle record of the PMM–to–PM relationship is in the structure of our framework, where the relationships between mapped objects are preserved. For example, there is a one–to–many relationship between PMM `ResourceTypes` and `Requirements`, and also between project `Resources` and `Allocations`, as shown in figure 6.3. In this way, there is a mapping not only of PMM objects to project objects, but also of PMM object relationships to project object relationships. We make use of this mapping of relationships in an implementation of the framework described in chapter 8, where we use project data to record the state of an active PMM.

6.3 Semantics of the Framework

In the previous sections we created mappings from the PMM model to the PM model. Semantic constraints are made on these mappings by the behaviour of their respective PMM classes. In the following sections we describe these semantics.

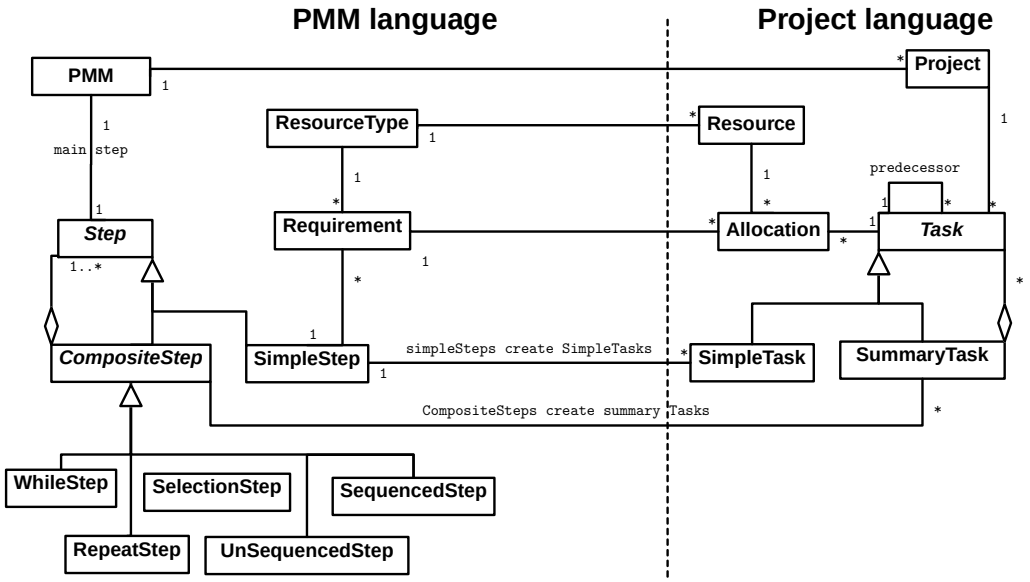


Figure 6.5: *Bringing all our PMM–to–PM mappings together: a conceptual model of our framework that links PMMs (left) to projects (right).*

6.3.1 PMMs ⇒ Projects

When a `PMM` is applied to achieve a particular `Goal`, a `Project` instance is created, and the PMM `main Step` then controls the production of the first Project *Task*.

6.3.2 PMM Steps ⇒ Project Tasks

As we showed in chapter 5, every PMM *Step* exhibits a certain behaviour when we record the Project *Tasks* that it creates. These behaviours are varieties of sequence, selection and repetition, as illustrated in table 6.1. Because *CompositeSteps* may contain *Steps* in an arbitrary nesting, the *Steps* so contained map to `Tasks` that reflect this nesting.

6.3.3 PMM Requirements ⇒ Project Allocations

A PMM `SimpleStep` may have `Requirements` for particular `ResourceTypes`. These `Requirements` are recorded as Project `Allocations` of `Resources` (of the specified `ResourceType`) to the *Task* associated with the PMM *Step*.

PMM *Steps*	Semantics of *Steps* contained	Map to Project *Tasks*
`SequencedStep`	Sequentially	A `SummaryTask` containing a sequence of Project *Tasks* that reflect the ordering of the set of *Steps* contained in the `SequencedStep`
`UnSequencedStep`	In parallel	A `SummaryTask` containing an unordered set of Project *Tasks* that reflect the set of *Steps* contained in the `UnSequencedStep`
`RepeatStep`	Repetitively. The number of repetitions is controlled by a test at the bottom of the loop	A `SummaryTask` containing a repeated sequence of Project *Tasks* that reflect the *Step* contained in the `RepeatStep`.
`WhileStep`	Repetitively. The number of repetitions is controlled by a test at the top of the loop.	A `SummaryTask` containing a repeated sequence of Project *Tasks* that reflect the *Step* contained in the `WhileStep`.
`SelectionStep`	One or no steps are selected, according to an expression in the `SelectionStep`.	A `SummaryTask` containing zero or one selected *Task*.
`SimpleStep`	Never contains *Steps*	One Project *SimpleTask*

Table 6.1: *The semantics of mapping different kinds of PMM* Step *to Project* Tasks.

6.3.4 PMM ResourceTypes ⇒ Project Resources

Many different algorithms exist for the allocation of resources to tasks; [75] describes 43 patterns, for example. We could use any of these patterns for assigning `Resources` to *Tasks*. For example: one very simple resource allocation pattern is: If a `Resource` of a given `ResourceType` exists in a Project then it is used for all allocations of the given `ResourceType`. If such a `Resource` does not exist in a Project instance then it is created and allocated.

6.3.5 Addressing Bussler's Mapping Problems

Having formulated our framework, we can test how well it addresses Bussler's list of mapping problems (table 2.1 on page 11). Table 6.2 summarizes how our framework concepts and semantics address the requirements of Bussler's mapping problems.

6.4 Extensibility of the Framework

The principle of separation of concerns provides us with powerful tools to extend the Framework: we can extend the PMM side with little or no effects on the PM side, and vice–versa.

6.4.1 PMM Extensibility

The complex behaviour of PMMs is confined to two areas: there is the semantics of PMM *Steps*, which act to constrain the structuring of Project *Tasks*, and there is the allocation of Project `Resources` of a particular PMM `ResourceType`.

The most likely part of the PMM model to require extension is the `Resource` allocation model. In section 6.3.4 we illustrated resource allocation with a very simple model: if a suitable `Resource` does not exist in a Project instance then it is created. Many more effective methods of resource allocation than this exist, and these allocations can be done in two basic ways: The allocation is made at the time a Project *Task* is created, or the allocation is deferred. Resource allocation could be made by some completely

WFMS/PMM concept	**Project Concept**	**Bussler's Framework Mapping**
Composite and elementary workflow	Task	SimpleSteps map to SimpleTasks, CompositeSteps map to SummaryTasks.
N/a	Tasks have duration	Duration of tasks is set in the PM domain.
Sequence	End-to-begin	Sequences of Steps map to Tasks with Predecessors.
Conditional Branching	N/a	Control flow of SelectionSteps determine the tasks that appear in the project.
Parallel Branching	Parallel Branching	UnSequencedSteps map to parallel Tasks.
Recursion	N/a	No recursion is available, but recursive algorithms could be changed to an in–line form.
Loops	N/a	WhileSteps and RepeatSteps map to sequences of Tasks.
N/a N/a	Begin–to–begin End–to–end	Begin–to–begin and end–to–end Task Predecessors can be specified in SequencedSteps.
Users, groups, roles, etc, together with complex assignment rules.	Resource	Assignment rules of arbitrary complexity can be applied because our framework maps ResourceTypes to Resources and Requirements to Allocations.
N/a	Resources have capacity and load	Resource attributes are visible to PMMs.
Data and Dataflow	N/a	Dataflow dependencies between PMM Steps can be mapped to Tasks with Predecessors.
Application integration	N/a	The "database level" connection of our framework can connect and integrate applications.

Table 6.2: *How our framework addresses Bussler's mapping of workflow management system (PMM) concepts to PM concepts, at the "database level".*

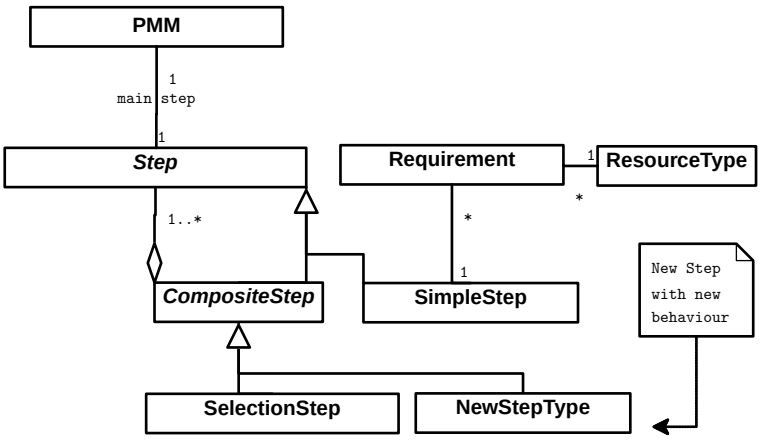


Figure 6.6: *Adding a new Step to the PMM model.*

separate process, using, for example: the allocation patterns that Russell [75] identifies.

One unfortunate consequence of mapping nested PMM *Steps* to nested Project *Tasks* might be the formation of parallel inheritance hierarchies [34]. However, our approach mitigates against the worst disadvantage of parallel hierarchies; that the creation of a new class in one hierarchy requires the creation of a new class in the other. Because all kinds of PMM *CompositeStep* map to Project `SummaryTasks`, we can add new *CompositeSteps* with new behaviour types to the PMM model, and not require the addition of new kinds of Project *Tasks*. Hence, the PM side of the Framework is unaffected by the addition of new kinds of PMM behaviour, and our framework has a clear path for such extensions, by adding new PMM *Step* types with new behaviours, as we illustrate in figure 6.6. For example: we have two kinds of PMM repetition step, although only one is strictly necessary. The `WhileStep` we define has a test at the top of the loop, whereas the `RepeatStep` has the test at the bottom. Either repetition step could be combined with a selection step to synthesize the behaviour of the other repetition step, but we avoid this complexity by defining both step types.

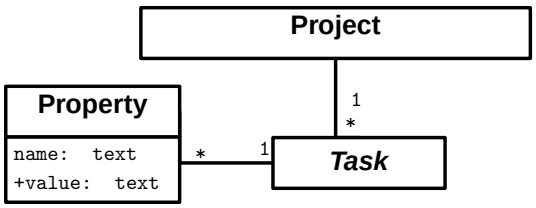


Figure 6.7: *Adding customizable properties to Tasks in the PM model.*

6.4.2 PM Extensibility

One of the more likely areas of extension to be required for the PM model is for more description of the PM objects: *Tasks*, `Resources` and `Allocations`. Our Framework already requires some attributes added to the basic PM model: we wish to record at least the PMM objects associated with Project objects, and we will discuss this more in section 7.1.2. Many PMM tools decorate the basic PM model with such concepts as Work Breakdown Structure. These additions are implemented as attributes added to the PM model, and some PM tools allow customizable attributes to be added to the basic descriptions of PM objects. Figure 6.7 illustrates the addition of customizable properties to Project *Tasks*, for example.

6.5 Conclusion

In this chapter, we have linked our PMM and PM models together in a framework that we propose as a solution to the research problem of chapter 2. The semantics of the PMM–to–PM mapping is defined by the behaviour of `PMMs`, *Steps*, `ResourceTypes` and `Requirements`. The Software Engineering principle of Separation of Concerns allows us to extend the frameworks PM and PMM models in a simple and logical way.

Chapter VII

Implementing the Framework

In chapter 6 we formulated UML classes and semantics that describe our framework at a conceptual level. In this chapter we will flesh out this conceptual description to the implementation level, and discuss different representations of our framework using three well–known meta–languages: unified modelling language (UML), extensible markup language (XML) and structured query language (SQL).

7.1 UML and object–oriented representation

We have used the Unified Modelling Language (UML) for the derivation of our framework. UML is the standard object modelling language, and it can be mapped to object–oriented languages such as Java. Figure 6.5 on page 60 is a conceptual, or high–level, UML model of the framework that must be fleshed out with attributes and methods to be implemented. These methods, and the attributes they require, can be derived from the semantics of the framework described in chapter 6, but first these semantics must be modelled. The next sections show how we applied an object–oriented pattern to do this.

7.1.1 Applying the Interpreter Pattern

In chapter 5, we used the Composite pattern ([36] pp 175–184) to formulate the PMM meta–model, and one result from this is that our PMM descriptions constitute a defining grammar for the Interpreter pattern ([36] pp 243–255). Figure 7.1 illustrates the interpreter pattern. This result provides a straightforward way of applying a PMM to a goal, by creating an Interpreter for our PMM data model.

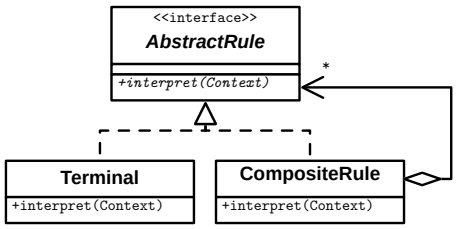


Figure 7.1: *The Interpreter pattern. A Composite pattern, as used for our PMM meta–model, can be used as the abstract syntax tree of an Interpreter ([36], p 255).*

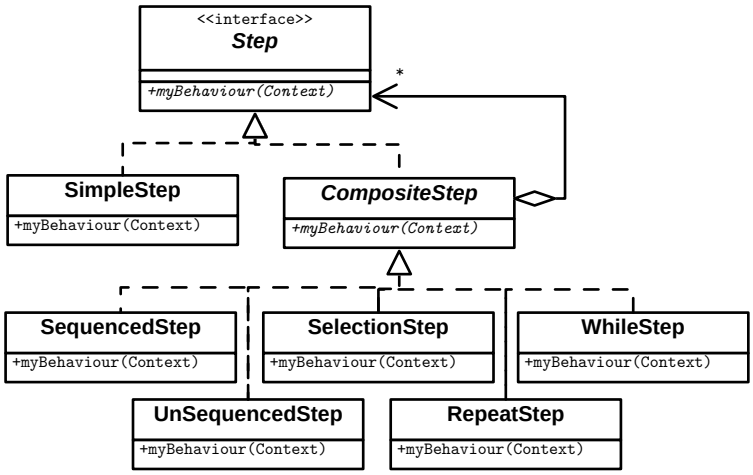


Figure 7.2: *The Interpreter pattern applied to the PMM meta–model. Polymorphism of the myBehaviour() method implements the different behaviours of the PMM Steps.*

Applying the Interpreter pattern to the behaviour of PMM *Steps* results in the model of figure 7.2. For this model, we have mapped Interpreter `CompositeRules` to PMM *CompositeStep*s, and the Interpreter `Terminal` to the PMM `SimpleStep`. The `interpret()` method from the Interpreter pattern is mapped to `myBehaviour()` methods for each kind of PMM *Step*. The context for the `myBehaviour()` method is provided by a `pTask` variable: this is a *Task* that will be the parent of every *Task* created by the `myBehaviour()` method.

A way to start the Interpreter is required, and this will happen when a

PMM is applied to a goal. We will create a PMM `applyToGoal()` method that, beginning with the PMM `main Step`, applies a PMM to a specified goal.

Finally, associated with `SimpleSteps` are one or more `Requirements` for `ResourceTypes`. To allocate Project `Resources` to *Tasks* mapped from the `SimpleStep`, we will create an `allocResource()` method for the `Requirement` class,

By applying the Interpreter pattern to the PMM model, we can perform a depth–first walk of the tree of *Steps* that describe a PMM, to invoke the semantics of a PMM applied to a goal. In appendix B, we describe in detail the methods required to implement the semantics of the framework, when applying a PMM to a goal.

7.1.2 Adding PMM Concepts To PM

In chapter 3 we derived a conceptual model of the basic common concepts in the PM domain that fits straightforwardly into the data models used by many common project tools (figure 3.6 on page 27). Our framework allows us to record PMM concepts in project data, and to do this we must augment our basic project data model. To make a project "PMM aware" in terms of our framework, it's necessary to augment our project data model with attributes to record the relationships crossing from the PMM side to the project side of the framework, as follows:

- the `Project` class gains `PMM` and `Goal` attributes, to record the PMM and goal for the project.
- The *Task* class gains a `taskType` attribute that relates it to the PMM *Step*.
- the `Resource` class gains a `resourceType` attribute.

This augmented project data could be stored in the data model of an existing project tool in a number of ways. For example: many project tools make available user–defined attributes, (or, alternatively: attributes for English–language descriptions) for projects, tasks and resources, that could be defined

for our purpose. If the project data model has no customization available at all, the augmented attributes could be stored in external data to record the PMM–to–project associations.

7.1.3 An Implementation–level OO Model of our Framework

If we augment the conceptual description of our framework with the PM attributes described above, and the methods and attributes described in appendix B, then we can create an implementation–level UML description of our framework—figure 7.3 on the next page illustrates. This description could be used to describe the classes and methods of a Java or similar object–oriented implementation of our framework.

7.2 A Relational Table Representation of the Framework

Although we have used the UML to design one implementation of our framework, we are not limited to an object–oriented implementation. We might have to implement the framework using a project tool that uses a relational database [13] to store project data, and the following sections describe how to do this.

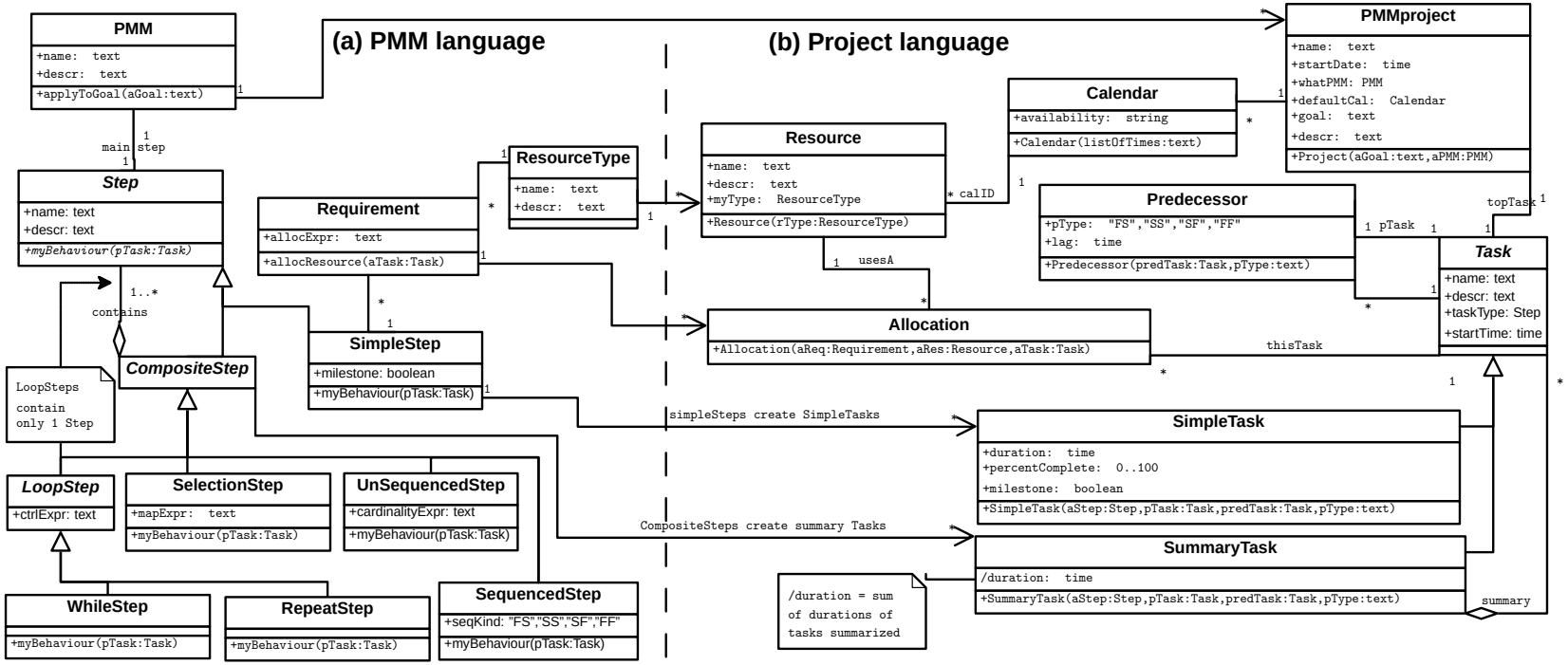
7.2.1 Mapping UML to Relational Tables

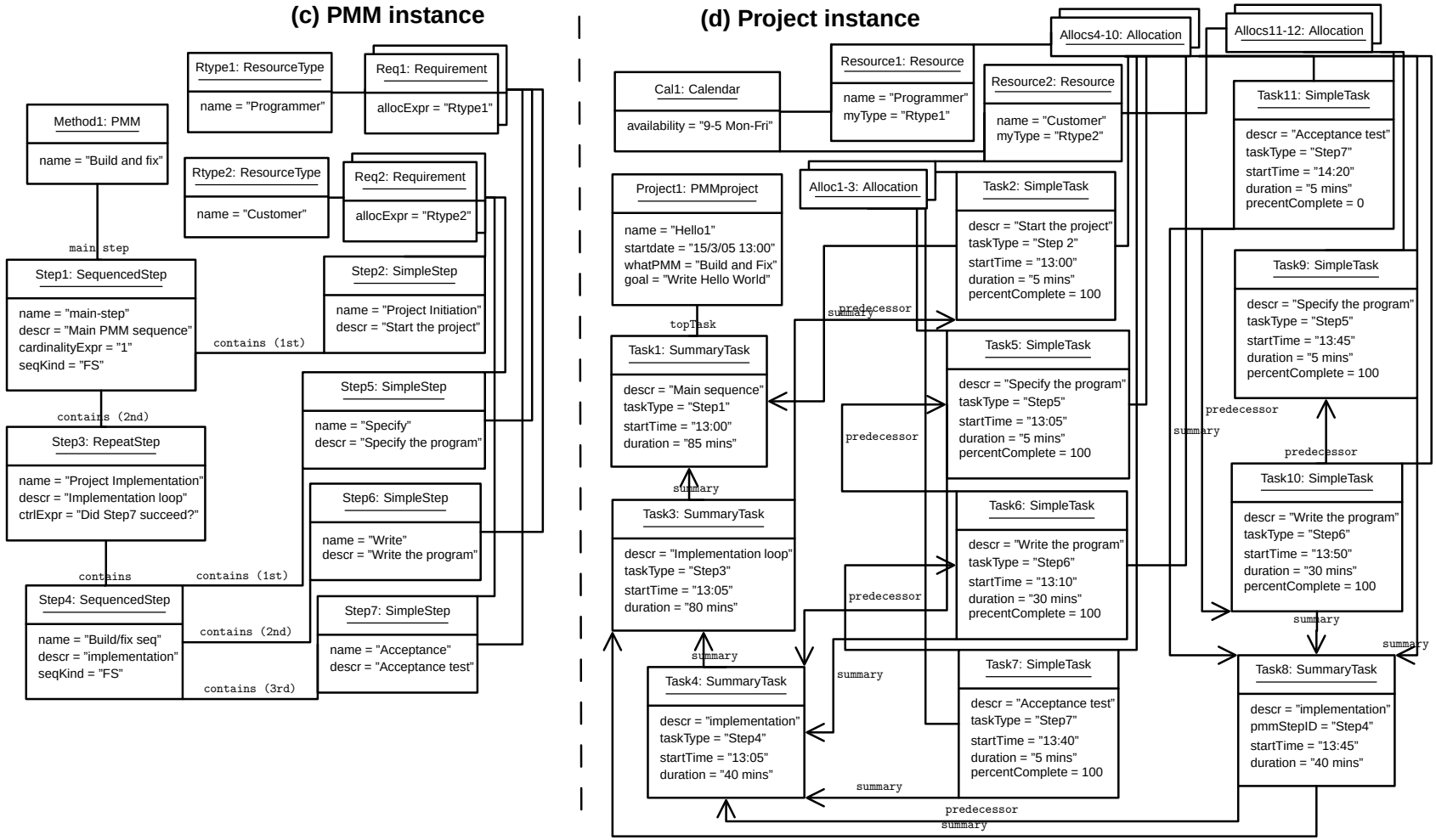
UML object–oriented data model concepts are a superset of the concepts represented with relational databases, but it is possible to map an object–oriented data model to relational tables by such techniques as using a "type" field to mimic inheritance. We mapped our UML framework description to relational tables as follows:

- Each type of PMM *Step* is assigned a separate table, whilst maintaining a unique *Step* ID across all step tables. This is to allow references to any step type to be unique, required by the polymorphism of steps in our PMM model. A similar strategy is used for project `SimpleTasks` and `SummaryTasks`.

- Each OO class attribute is a field in the table representing the class,

Figure 7.3: *An implementation–level UML diagram of our framework, including methods and attributes, that links PMMs (a) to projects (b). In the bottom left half are example UML Object diagrams of the Build and Fix PMM (c), and a corresponding project instance (d) created using Build and Fix.*




(c) PMM instance
Method1: PMM
name = "Build and fix"
Rtype1: ResourceType
name = "Programmer"
Req1: Requirement
allocExpr = "Rtype1"
Rtype2: ResourceType
name = "Customer"
Req2: Requirement
allocExpr = "Rtype2"
main step
Step1: SequencedStep
name = "main-step"
descr = "Main PMM sequence"
cardinalityExpr = "1"
seqKind = "FS"
Step2: SimpleStep
name = "Project Initiation"
descr = "Start the project"
contains (1st)
contains (2nd)
Step3: RepeatStep
name = "Project Implementation"
descr = "Implementation loop"
ctrlExpr = "Did Step7 succeed?"
Step5: SimpleStep
name = "Specify"
descr = "Specify the program"
Step6: SimpleStep
name = "Write"
descr = "Write the program"
contains
contains (1st)
Step4: SequencedStep
name = "Build/fix seq"
descr = "implementation"
seqKind = "FS"
contains (2nd)
Step7: SimpleStep
name = "Acceptance"
descr = "Acceptance test"
contains (3rd)
(d) Project instance
Cal1: Calendar
availability = "9-5 Mon-Fri"
Resource1: Resource
name = "Programmer"
myType = "Rtype1"
Resource2: Resource
name = "Customer"
myType = "Rtype2"
Allocs4-10: Allocation
Allocs11-12: Allocation
Project1: PMMproject
name = "Hello1"
startdate = "15/3/05 13:00"
whatPMM = "Build and Fix"
goal = "Write Hello World"
Alloc1-3: Allocation
Task2: SimpleTask
descr = "Start the project"
taskType = "Step 2"
startTime = "13:00"
duration = "5 mins"
percentComplete = 100
Task11: SimpleTask
descr = "Acceptance test"
taskType = "Step7"
startTime = "14:20"
duration = "5 mins"
precentComplete = 0
Task9: SimpleTask
descr = "Specify the program"
taskType = "Step5"
startTime = "13:45"
duration = "5 mins"
percentComplete = 100
topTask
predecessor
summary
Task1: SummaryTask
descr = "Main sequence"
taskType = "Step1"
startTime = "13:00"
duration = "85 mins"
Task5: SimpleTask
descr = "Specify the program"
taskType = "Step5"
startTime = "13:05"
duration = "5 mins"
percentComplete = 100
Task10: SimpleTask
descr = "Write the program"
taskType = "Step6"
startTime = "13:50"
duration = "30 mins"
percentComplete = 100
Task3: SummaryTask
descr = "Implementation loop"
taskType = "Step3"
startTime = "13:05"
duration = "80 mins"
Task6: SimpleTask
descr = "Write the program"
taskType = "Step6"
startTime = "13:10"
duration = "30 mins"
precentComplete = 100
Task4: SummaryTask
descr = "implementation"
taskType = "Step4"
startTime = "13:05"
duration = "40 mins"
Task7: SimpleTask
descr = "Acceptance test"
taskType = "Step7"
startTime = "13:40"
duration = "5 mins"
percentComplete = 100
Task8: SummaryTask
descr = "implementation"
pmmStepID = "Step4"
startTime = "13:45"
duration = "40 mins"

so that the `SimpleStep.name` attribute maps to a `name` field in the `SimpleStep` table.

- Relationships are represented by an attribute in the table, eg: the `parentStepID` field is used to represent the parent–child relationship of PMM steps.

- The `SequencedStep` table has an `ordering` field that is used to order the child steps, because relational queries do not necessarily preserve any kind of row ordering.

We designed relational tables, shown in tables 7.1 and 7.2, to represent our framework. We have populated the tables with data to represent the Build and Fix PMM, and a project instance created from this PMM.

7.2.2 Procedural Code for the Relational Representation

Relational tables cannot store any of the semantics that we describe in appendix B. These must be implemented as program code that manipulates the tables. The code is procedural, not object–oriented, and so object–oriented patterns such as the Interpreter are not directly applicable to procedural code. However, we can use strategies to map the object–oriented framework semantics to procedural pseudocode that manipulates relational tables, as follows:

- Object initializers are mapped to a procedure that is used whenever a new row is created in the table to which we mapped the object class.

- Object methods are mapped to procedures that manipulate the relational tables.

We used the above strategies to create pseudocode that applies a selected PMM to create a project instance, for a specified goal, listed in appendix B.3. This pseudocode was desk–checked, using the PMM in table 7.1 to create the project instance in table 7.2.

Project Management Method

id	name	description	mainStepID
1	Build and Fix	Simplest possible PMM	1

SequencedStep

id	name	description	seqKind	ordering	parentStep
1	main step	main seq	"FS"	2,3	null
4	build/fix seq	inner loop seq	"FS"	5,6,7	3

SimpleStep

id	name	description	milestone	parentStep
2	Initiation	Project setup activities	false	1
5	Specify	Specify the program	false	4
6	Implement	Write the program	false	4
7	Accept	Customer Acceptance Test	false	4

RepeatStep

id	name	description	ctrl expression	parentStep
3	Implementation	impl loop	Did step 7 pass?	1

ResourceType

id	name	description
1	Programmer	Writes programs
2	Customer	Sets requirements

Requirements

simple–StepID	**resource–TypeID**
2	1
2	2
5	1
5	2
6	1
7	1
7	2

Table 7.1: *Relational tables to represent the Build and Fix PMM, equivalent to the UML implementation diagram of figure 7.3(a). The data in the tables is equivalent to the UML object diagram of figure 7.3(c). We omit the unused relational tables for WhileSteps, UnSequencedSteps and SelectionSteps that are similar in form to the SequencedStep and RepeatStep tables shown above.*

PMMproject

id	name	descr	main Task	startdate	Default cal	what PMM	goal
1	Hello	A test project	1	August 22, 2007 1pm	1	Build & Fix	Create "Hello World"

SummaryTask

id	name	descr	startTime	duration	parent	taskType
1	main-step		13:00	85 mins	—	1
3	Project Impl		13:05	80 mins	1	3
4	Build/Fix seq		13:05	40 mins	3	4
8	Build/Fix seq		13:45	40 mins	3	4

SimpleTask

id	name	descr	start Time	duration	parent ID	% com-plete	mile stone	task Type
2	Project initiation		13:00	5 mins	1	100	false	2
5	Specify program		13:05	5 mins	4	100	false	5
6	Write program		13:10	30 mins	4	100	false	6
7	Acceptance Test		13:40	5 mins	4	100	false	7
9	Specify program		13:45	5 mins	8	100	false	5
10	Write program		13:50	30 mins	8	100	false	6
11	Acceptance Test		14:20	5 mins	8	100	false	7

Predecessor

taskID	**predecessor taskID**	pType	lag
3	2	"FS"	0
6	5	"FS"	0
7	6	"FS"	0
8	4	"FS"	0
10	9	"FS"	0
11	10	"FS"	0

Resource

id	name	descr	calendarID	resourceType
1	Programmer	Ace Computer Services	1	1
2	Customer	Acme Inc	1	2

Allocation

thisTask	**usesA**
2	1
2	2
5	1
5	2
6	1
7	1
7	2
9	1
9	2
10	1
11	1
11	2

Calendar

id	availability
1	9am to 5pm Mon–Fri

Table 7.2: *Relational tables equivalent to the UML implementation diagram of figure 7.3(b). The tables contain a project instance created with the Build and Fix PMM of table 7.1, equivalent to the UML object diagram of figure 7.3(d).*

UML construct	**XSD entity**
Classes	An <xsd:complexType> that defines an <xsd:element>
Attributes	An <xsd:attribute>
One-to-one & one-to-many associations	<xsd:element> groups contained in <xsd:element>s. Alternatively, association attributes or association elements may be used to record the ID of associated elements. The association elements may be contained in a root-level element, or in one of the elements of the association.
Many–Many Relations	Association elements, as above.
Generalization (Inheritance)	The <xsd:complexType> of the parent class is used as the base type of an <xsd:extension> in an <xsd:complexType> that defines the child elements (classes). An xsd:choice construct is used to refer to the child elements in an abstract way.
Abstract classes	An <xsd:complexType abstract=true> is used to define a base type for use with <xsd:extension>.
Enumerated types	<xsd:enumeration> values for an <xsd:restriction> on a base type.
Methods	No direct representation. Methods are implemented as program code associated with objects that are mapped from XML data.

Table 7.3: *Mapping the UML to XML Schema (XSD) representation.*

7.3 An XML Representation of the Framework

XML [93] is an important and widely–used data representation that, for example, is used by the Planner and GanttProject PM tools for their native data storage format, and Microsoft Project can export XML data, as we described in chapter 3. We can map UML classes to XML, as proposed in [29] for instance. Table 7.3 lists the mappings we used when creating the "TPMM" XML schema ([21], listed in appendix C), from the UML description of figure 7.3(a).

Using these mappings, OO classes such as `SimpleStep`, `RepeatStep` and

OO Classes	**XSD entities**
Step	<typeBaseStep> abstract type
SimpleStep	<SimpleStep> element
Step.name	name attribute of the <typeBaseStep> abstract type
Task	<typeBaseTask> abstract type
SimpleTask SimpleTask.name	<SimpleTask> element name attribute of the <SimpleTask> element

OO relationships	**XSD relationships**
SimpleStep to Requirement	<SimpleStep> containing a <Requirements> element, containing >= 1 <Requirement> elements etc...
SimpleStep and WhileStep generalize to abstract *Step*	<typeBaseStep> abstract type generalizes <typeSimpleStep> and <typeWhileStep> that define <SimpleStep> and <WhileStep> elements.

Table 7.4: *How we mapped some of the classes and relationships from our framework of figure 7.3 to an XML Schema description.*

ResourceType were transliterated to XML elements of the same name. OO attributes such as SimpleStep.name were transliterated to an XML name attribute attached to the <SimpleStep> element. OO relationships such as the one–to–many relationship from SimpleStep to Requirement were implemented by nesting inside the <SimpleStep> element a <Requirements> element. This element contains one or more <Requirement> elements. OO generalization of PMM *Step* types was transliterated to a <typeBaseStep> that had the abstract attribute set. An <xsd:extension> was then applied to the <typeBaseStep> to produce a <typeCompositeStep> that could contain one or more <xsd:group> elements that contained the PMM step types generalized. The <typeCompositeStep> is also abstract. It was further extended to final types <SequencedStep> and <SelectionStep>. We also created an abstract <typeUnaryStep> that could contain one and only one PMM step, and was extended to final types <RepeatStep> <WhileStep> and <UnSequencedStep>. Table 7.4 summarizes some of these mappings. In a similar way, we can map the PM data model of figure 7.3(b) to an XML schema for project data.

The TPMM schema closely mirrors the OO classes, attributes and relationships from which it was derived, and it is quite prescriptive, so that a <WhileStep> is allowed to contain one and only one of the steps extended from <typeBaseStep>, for example. A simpler and more permissive XML schema or DTD might allow any number of steps to be so contained. The resultant XML may be ambiguous and difficult to interpret, however. We used TPMM to write PMM descriptions for a prototype implementation of our framework that is described in chapter 8.

7.3.1 Semantics for XML data

Similarly to a relational representation, the XML schemata above do not include the semantics our framework requires, and they must be implemented as program code. One way of implementing these semantics would be use the strategy suggested in table 7.3, and map the XML data to objects that have methods associated with them in some object–oriented language. A technology such as the Java Architecture for XML Binding (JAXB [45])

would be suitable for such an implementation. The object–oriented methods of section B.1 could then be applied to objects representing the XML data of PMMs and projects.

Another way of implementing the semantics of our framework is to take a much more "XML–oriented" approach. In this case, the data is manipulated in a way much closer to the logical structure of an XML tree of nodes with attributes. Such an approach relies on procedural code to manipulate the XML data, similarly to the approach used for relational tables in section 7.2.2, but rather than creating rows in tables and fields in rows, we create elements and attributes in a tree of XML data. This XML–oriented approach is simple to implement because our framework is based on the intrinsically tree–structured Composite pattern, and so the XML tree structure quite easily reflects the relationships in both project and PMM data. For an implementation, we require some fairly direct interface to the XML data such as Java/DOM [27] that provides us with methods for creating and manipulating XML elements and attributes. We used this strategy for an implementation described in chapter 8.

7.4 Conclusion

In this chapter, we have produced three different implementation–level descriptions of our framework, using UML, relational tables and XML. This is possible because our framework is independent of any particular representation. In the next chapter, we will describe a Java/XML–based implementation of the framework, and also some visualizations of PMM and project data. These applications of our framework use the data–level linkage that it makes between PMMs and projects.

Chapter VIII

Applying the Framework to some Real Problems

In chapters 1 and 2, we outlined some of the difficulties raised, and opportunities afforded, by the application of PMMs to projects. In this chapter we first describe some specific problems and opportunities raised by project managers, during discussions about the application of PMMs to projects. Our framework supports the creation of tools and visualizations that we will then describe, with examples, to address these problems.

8.1 Some Real–World Problems with Projects and PMMs

During the course of this thesis, we informally discussed project management and PMMs with IT project managers who had 10 or more years of experience, including industry practitioners from small and large enterprises, project management tutors and academics. A half–dozen or so project managers raised issues for which our framework could support software tools. These included:

- Project generation and scenario or "what–if" analysis is laborious with project tools.

- The complexity of PMMs makes them difficult to describe and follow.

- PMMs are difficult to apply to projects, because of the mismatch between PMM and project data. For example: the potentially iterative Waterfall method is still often used in IT projects, but as a strictly once–through project template, rather than a PMM. The "software project" template included with Microsoft Project works similarly.

- When PMMs are applied to projects, they have to be "hidden" in the project data. For example: a project manager we spoke to assigned one long project task for the operation of an iterative PMM, so that the tasks generated by the PMM were not recorded (figure 8.1). Similarly, the code development part of the the software development template packaged with Microsoft Project 2003 consists of one task that takes three weeks. Compare this code development task to, say: the application of Extreme Programming (XP) to a software development project. For each software feature, XP would require the code development to run through at least one, and probably several, code/test cycles, each of which consists of several tasks that would probably be in the region of hours or days, rather than weeks. The Microsoft Project template glosses over this complexity.

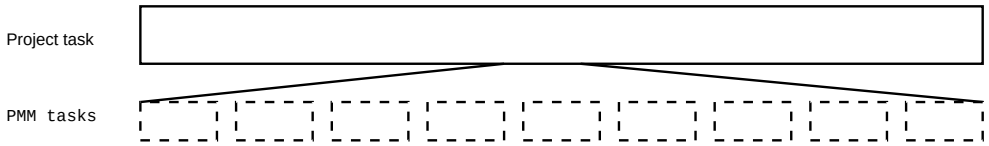


Figure 8.1: *Hiding PMM tasks in a project task—only one project task is recorded because of the difficulty of recording PMM information with conventional project tools.*

8.2 Project Generation and Scenario Analysis

One of the more laborious aspects of using the PM tools we studied is the creation of a project plan. When creating a project plan from scratch, the project manager is required to manually enter project task information, the relationships between tasks, the project resource information and finally assignments of resources to tasks. Any revision to the plan requires manual alterations in a similar way. Scenario analysis, involving the generation of project data for many different situations, multiplies this labour.

There is recognition of these problems, and some PM tools have facilities available to automate project creation: Some project tools can import

resource information from, eg: an LDAP [48] directory server maintaining employee information for a company, if such is available. Template plans are also available for a limited range of project types—but as we have pointed out in chapter 3, templates do not assist with the revision of a plan.

8.2.1 Project Mentor

To address the problem of easily generating project plans, and especially the problems of applying a PMM to a project, we wrote a Java applet; "Project Mentor" [20], that uses the data framework of figure 7.3 on page 70, implemented as XML schemata. Designing Project Mentor was also a good test of how implementable our framework actually is. Project Mentor uses both the PM and PMM sides of our framework, and also the semantic connection between them. The appearance of Project Mentor is shown in figure 8.5 (at the bottom—the visualization on the top will be described in section 8.3). Project Mentor applies a PMM to a specified goal, and automatically generates Projects, tasks, resources and assignments under the control of a project manager.

In section 7.3 on page 76, we described how we mapped our framework to XML schemata. PMM descriptions for Project Mentor are written with the XML schema TPMM, and we used this schema to write a small library of PMM descriptions, including the Waterfall method [73], Extreme Programming [5] and the Rational Method [71]. These PMMs, available at [20], were used to generate projects with Project Mentor.

For recording the "PMM aware" project data described in section 7.1.2 on page 68, we could have mapped the PM model of our framework to a custom XML schema, but we chose instead to use an existing project XML DTD that closely matches our frameworks PM model. Using an existing DTD allows Project Mentor to generate "PMM aware" project data that can be viewed and manipulated with an existing software tool, even though this tool was not designed with reference to any PMM concepts. The data–level connection between PM and PMM data that our framework makes this possible. Appendix C.2 lists this DTD, used in the Planner project tool [42], and describes how we augmented it with PMM attributes, as per

section 7.1.2. There are some disadvantages with using the Planner DTD. DTDs do not provide abstract types and inheritance, for example, so the data model as implemented is less descriptive then an XSD description we could have derived from our UML PM model. Also, there are specific problems with the Planner DTD such as the ambiguity of the <property> element. However, the advantage of using an existing tool for dealing with project data outweighed these minor problems.

We used the "XML–oriented" approach of section 7.3.1 to implement the PMM semantics of chapter 6 as Java code, that interacts with PMM and PM data stored in DOM [27] trees reflecting the above XML schemata. The Java code is procedural code that directly manipulates these DOM trees.

When started, Project Mentor first queries for the location of a project instance. If the project does not contain the location of a PMM then a PMM is queried for, and the PMM is applied from its initial step to a goal that the user specifies. If the project does contain the location of a PMM, then Project Mentor loads this PMM description and reconstructs the state of the PMM process from the project data, then continuing with the application of the PMM. This is possible because the relationship between PMM and PM data in our framework is sufficiently close that Project Mentor can make use of the project data it produces as an XML serialization of the PMM process state. To record the process state, we indicate which project tasks are active in terms of the PMM[1]. Project Mentor may be interrupted at any time, by typing "q" in response to a query from the applet. The project instance created up to that point can later be used as the initial project instance for starting Project Mentor, which will then carry on applying the PMM to the goal originally specified by the user, and stored in the project instance. When Project Mentor terminates, either by being interrupted or when a PMM has finished, project data is loaded into a web browser window by the applet to be saved for viewing or manipulation with Planner.

Project Mentor allows the application of PMMs to projects, but it still relies on project management tools to deal with project–level attributes, such as task length: Project Mentor assigns a default task length of one day, and

[1] In the Planner DTD, we do this by setting the task priority to 9999

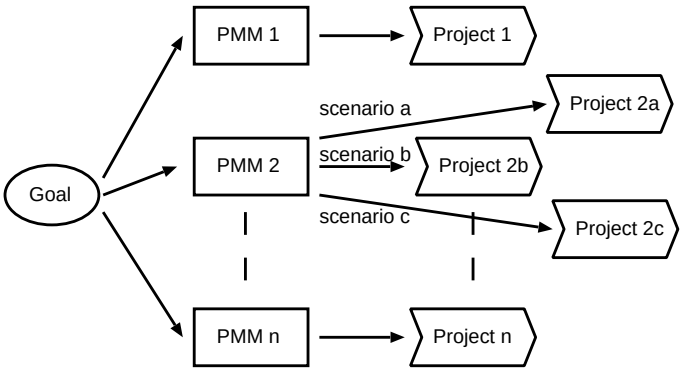


Figure 8.2: *Scenario analysis: Generating different project instances for the same goal, using different PMMs and control flows. Each PMM can be applied to many different scenarios (in this case, PMM 2 is applied three different ways), by operating the control flow of the PMM in different ways.*

relies on the use of the Planner project tool to customize task length. One alternative way of setting task length might be to add a `taskLength` attribute to the PMM `SimpleStep` class, that was then used to set the length of every project `Task` generated from the `SimpleStep`. This approach could fit in with the strategy employed by such PMMs as Extreme Programming. XP has hard deadlines pre–set in its steps, such as a regular (say) fortnightly release cycle.

Scenario Analysis

Project managers can make use of the powerful features of both PMMs and project management tool, for easily generating project instances when, eg: analysing many different project scenarios. Scenario analysis using a project–generation tool like Project Mentor is illustrated in figure 8.2. Using a library of PMMs with Project Mentor, generation of projects for many different PMMs and project scenarios becomes possible.

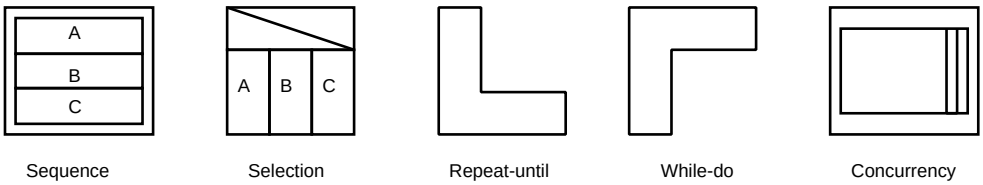


Figure 8.3: *How we applied the elements of Nassi–Shneiderman diagrams to represent from left to right: a SequencedStep, SelectionStep, RepeatStep, WhileStep and UnSequencedStep.*

8.3 Process Visualization

To address the problem of comprehending PMMs and their application, we investigated their visualization and animation. Project schedules, tasks and the assignment of resources to tasks can be visualized in a project instance using, for example: the Gantt chart display of a PM tool. However, the process control flow that PMMs use is quite abstract and difficult to visualize, and it is more dynamic than a project plan. For example: our Project Mentor implementation appears to the user as a "black box", that asks questions and then produces a project instance. Software engineers have similar problems when trying to understand computer programs (Dijkstra [25] observed in 1968 that "our intellectual powers are rather geared to master static relations and that our powers to visualize processes evolving in time are relatively poorly developed"), and systems have been created to visualize and animate program control and data structures (eg: [69][72][62]). Wiggins [90] reviews many of these systems, and concludes that "While it seems to be the consensus that animations are helpful, as yet no study has substantiated that fact". It may be that visualization and animation of PMMs would be useful for project managers, but we could find little written about this topic, and so we investigated ways of visualizing and animating PMMs and PMM–driven projects with process visualizations.

Much of the descriptive capabilities of a PMM lie in its control flow structures. To visualize this control flow, we examined some methods of visualizing process control flow, such as flowcharts and the related UML action diagrams, etc. One example visualization is that of chapter 5, that we used

to track the progress of a PMM and illustrate control sequences. Our simple visualization quickly becomes quite bulky when tracking a large number of steps, but we found that Nassi–Shneiderman (NS) diagrams [59] can provide a compact visualization of PMMs and PMM activity. NS diagrams are a well–known visualization of process control structures that map straightforwardly to the PMM control structures we define, as shown in figure 8.3. We applied NS diagrams to our simple implementation. First, we created NS representations of the PMMs we had described. An XSL transform, "pmmtosvg.xsl" [23], was written to convert PMMs described using our TPMM schema to SVG [83] graphics files representing a NS diagram of the PMM, as figure 8.4 illustrates. An SVG viewer, such as a web browser with an SVG plugin, can then view the NS diagrams.

Next, we tried animating the NS diagrams to illustrate the flow of control in a PMM. As each step is applied, that step is highlighted in a NS diagram of the PMM. This animation was created by making pmmtosvg.xsl produce, in addition to the basic NS diagram of a PMM, an NS diagram for each PMM step, with that step highlighted. These files are used by our Project Mentor Java applet when it is passed a parameter; "ANIMATE". When animating a PMM, the applet pops up a browser window and loads SVG files produced by pmmtosvg.xsl into the window, for each PMM step as it is applied. Figure 8.5 illustrates one step of the Waterfall PMM being animated using an NS diagram in this way.

8.4 Visualizing PMM Concepts in Project Data

Section 8.3 described an animation of control flow for PMMs. Visualization of PMM control flow in project data might also be useful for project managers, so that the pattern of application of the PMM in the project data can be made obvious. Some conventional workflow systems have the ability to export tasks to project tools, but when this is done, process information is lost, so that it is impossible to relate the project data to the process that produced it. Using our framework, a project instance can be augmented with attributes indicating which PMM step each project task was produced by. The project data then acts in an analogous way to the execution trace of a computer

```
<PMM name="Build And Fix">
  <SequencedStep name="main-step" seqKind="FS" id="1">
    <SimpleStep name="Project Initiation" id="2"/>
    <RepeatStep name="Project Implementation" id="3"
      ctrlExpr="Customer Acceptance Test outcome">
      <SequencedStep name="Build/fix seq" seqKind="FS" id="4">
        <SimpleStep name="Specify the program" id="5"/>
        <SimpleStep name="Write the program" id="6"/>
        <SimpleStep name="Customer Acceptance Test" id="7"/>
      </SequencedStep>
    </RepeatStep>
  </SequencedStep>
</PMM>
```

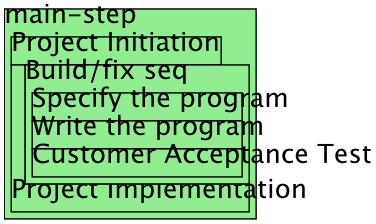
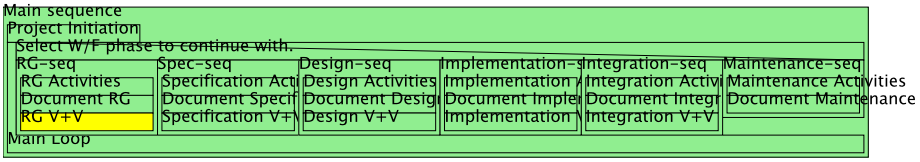


Figure 8.4: *Moving from an XML representation of a PMM (simplified for illustration) to a Nassi–Shneiderman (NS) diagram representing the same PMM steps. We created the above Nassi–Shneiderman diagram by using batik [4] to render an SVG graphic, that was produced by applying pmmtosvg.xsl to the XML file with our description of the Build And Fix PMM.*



Project Mentor

What is the URI of a Project Instance?
What is the URI of a PMM?
What is the Goal of this project?
Step (live): main-step
Step (live): Project Initiation
Step (live): Requirements Gathering
Step (live): RG-seq
Step (live): RG Activities
Step (live): Document RG
Step (live): RG V+V
Redo Requirements Gathering: Loop again (Y/N)?

N

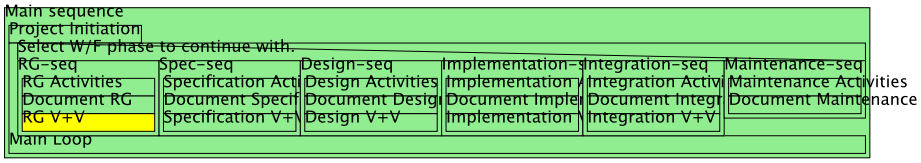
Figure 8.5: *The Project Mentor applet (bottom) using a Nassi–Shneiderman diagram [59] (top) to visualize the application of the Waterfall method. PMM steps in the Nassi–Shneiderman diagram are highlighted to illustrate the active step.*

program, and it becomes possible to visualize the history of the application of a process, as we examine project data. For example: we could highlight the steps in a PMM visualization that correspond to project tasks.

To test the feasibility of this approach, we made more use of the NS diagrams described in section 8.3. We altered two of the XSL transforms (html1_gantt.xsl planner2html.xsl, available at [22]) that Planner version 0.13 uses to export project data as HTML files. We added Javascript [30] to the HTML files to pop up an extra browser window that displays SVG files produced by pmmtosvg.xsl. As the mouse is moved over task names in the Gantt chart of project data on the HTML page, corresponding PMM steps in a Nassi–Shneiderman diagram are highlighted. Figure 8.6 illustrates.

8.5 Creating And Editing PMMs

Our framework affords the possibility of creating tools that manipulate PMM descriptions. For example: because we can represent our PMM language as an XML schema, it is possible to use one of the many available XML schema–sensitive editors [65] such as Xeena [43] to create and revise PMMs. When writing PMMs for Project Mentor, our personal preference is to use the XML mode of gvim[64], but a schema–sensitive editor constrains the user to only adding XML tags that are allowed by an XML schema, in the context the user is editing. Such general–purpose editors are very XML–oriented, so that the user is often dealing directly with XML tags, in some kind of tree–based view (because the basic architecture of XML documents is a tree). For editing PMMs, we could create a graphical tool based on, say: Nassi–Shneiderman diagrams, using a direct–manipulation interface. The user would manipulate a NS diagram representing the PMM they wished to edit, while the tool dealt with the PMM data. Such a tool would relate closely to the NS–based visualizations we previously described, so that if a project manager found it necessary to "tweak" a PMM–generated project by adding or changing tasks (something that we will see does happen, in chapter 9), the visualization of section 8.4 would highlight exactly which area of the PMM these changes related to. The PMM could then be revised to reflect the project tweaking, as appropriate.



Unnamed Project

Start: April 10, 2006
Finish: April 12, 2006
PMM: file:/home/cosc/staff/ajd41/.netbeans/work/pmm/Waterfall.xml
Goal: test
Report Date: April 11, 2006

Gantt Chart

WBS	Name	Work	Week 15, 2006 10 11 12 13 14 15 16	Week 16, 2006 17 18 19 20 21 22 23	Week 17, 2006 24 25 26 27 28 29
1	**main-step**	**3d**			
1.1	Project Initiation	1d	Cust		
1.2	**Requirements Gathering**	**2d**			
1.2.1	**RG-seq**	**2d**			
1.2.1.1	RG Activities	1d	Cust, Syst		
1.2.1.2	Document RG	1d	QA T		
1.2.1.3	RG V+V	1d	Cust, QA T		

Tasks

WBS	Name	Start	Finish	Work	Priority	Complete	Cost
1	**main-step**	Apr 10	Apr 12	3d			
1.1	Project Initiation	Apr 10	Apr 10	1d		0%	
1.2	**Requirements Gathering**	Apr 11	Apr 12	2d			
1.2.1	**RG-seq**	Apr 11	Apr 12	2d			
1.2.1.1	RG Activities	Apr 11	Apr 11	1d		0%	
1.2.1.2	Document RG	Apr 11	Apr 12	1d		0%	
1.2.1.3	RG V+V	Apr 12	Apr 12	1d		0%	

Resources

Name	Short name	Type	Group	Email	Cost
Customer	Cust	Work			0
QA Team	QA T	Work			0
Systems Analyst	Syst	Work			0

This file was generated by Planner

Figure 8.6: *Visualizing PMM "footprints" in project data. The "RG V+V" PMM step is highlighted as the user mouses over a corresponding task in a Gantt chart of the project. The Gantt chart was created as an HTML page, using planner2html.xsl.*

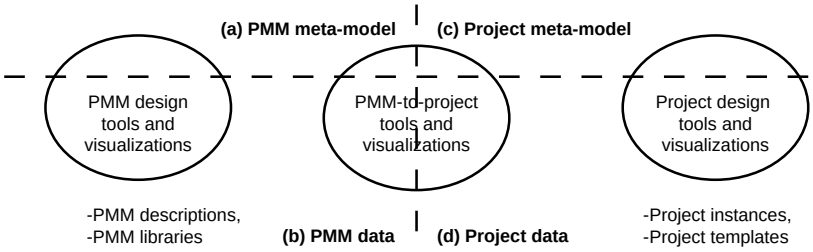


Figure 8.7: *How tools and visualizations fit into the meta–models and data in our framework of chapter 6.*

8.6 Conclusion

In this chapter, we have listed some problems found with conventional project management, especially when PMMs are applied to projects. We have then illustrated, with examples, how our framework affords the creation of tools and visualizations that address these problems. To recap:

- The difficulty of project generation and scenario analysis is addressed in section 8.2.
- The complexity of PMMs is addressed in sections 8.3, 8.4 and 8.5.
- The difficulties of applying PMMs to projects are addressed in sections 8.2, 8.3 and 8.4.

Figure 8.7 uses our framework to categorize the tools and visualizations we have described in this chapter:

- Data itself may be usable as a tool, such as PMM descriptions (b) and project templates (d).
- Software project tools such as Microsoft Project cross quadrants (c) and (d), since they produce project data constrained by a meta–model.
- Our Project Mentor tool of section 8.2 similarly uses quadrants (c) and (d), and also quadrants (a) and (b), because it makes use of PMM descriptions.

- The PMM editors discussed in section 8.5 and the process visualization of section 8.3 are described by quadrants (a) and (b).

- The project data visualization of section 8.4 uses quadrants (b) and (d).

In chapter 9 we will describe a case study and a survey that tests our framework against some extant project data.

Chapter IX

A Case Study and a Survey

In this chapter, we present two tests we applied to explore how useful the concepts in our framework might be to project managers.

The test, and the way that we chose, is to examine some published project plans, and look for features that indicate the application of PMM ideas that we can relate to our framework.

First, we describe a case study that compares how a project manager applies a PMM to a goal with how our Project Mentor implementation similarly applies a PMM. Next, we describe a survey that attempted to find PMM features in project data, what level of project complexity project managers normally work with, and any features that indicate the application of PMM ideas we can relate to our framework.

9.1 Comparing Human and PMM–Generated Project Plans

To show how our framework provides a context for the analysis of a project in terms of a PMM, and to illustrate some of the difficulties encountered by project managers when applying a PMM to a project, we undertook a case study to compare a PMM–generated project plan to a project plan created by a human project manager, who used a similar PMM.

9.1.1 Method

We used our framework as the context for a comparison of PMM features in an existing plan for an IT project, created with reference to the Rational Unified Process (RUP), to a plan generated with Project Mentor, also using RUP and with a similar goal. We observed how many tasks in the PMM–generated plan were present in the human–generated plan, and in what order, how the

project resources were assigned to the tasks, and the general descriptiveness of the project plans.

For the case study, we created a description of the RUP PMM with reference to an IBM/Rational "RUP best practices" document [71]. The human–generated plan was examined and tasks were manually classified according to the steps in our RUP PMM. For instance: RUP calls for phases of Inception, then Elaboration, Construction and Transition, and so we looked for major tasks with similar names and the above ordering in the project plan.

We then used our RUP description and Project Mentor to generate a project plan with a similar goal and control flow to the human–generated plan. For example: there were no code–writing iterations in the human–generated plan, although an "iteration planning" task was included, therefore we did not create any iterations in the generated plan. We compared the PMM–generated plan to the human–generated plan, and compared the plans for the existence of tasks mandated by the PMM, and the ordering of the tasks.

9.1.2 Observations and Discussion

Figure 9.1 illustrates how we went about the comparison described above: Where equivalent tasks (by the author's reckoning) were present in the human generated plan (a), we have marked the tasks in the generated plan (b) with an asterisk and the task name from plan (a). Some tasks in plan (a) were in a different part of the plan than our RUP PMM specified, and we have annotated these in plan (b) with the name of the RUP phase from plan (a) that they came from.

Tasks

Classification of the tasks in the human–generated plan was relatively straightforward because the task names corresponded well to the step names in our RUP PMM (our Project Mentor implementation defaults to naming project tasks with their corresponding PMM step name for similar reasons of clarity). All of the PMM–required tasks classified in the human–generated plan could be mapped to our PMM–generated plan, but the human–generated

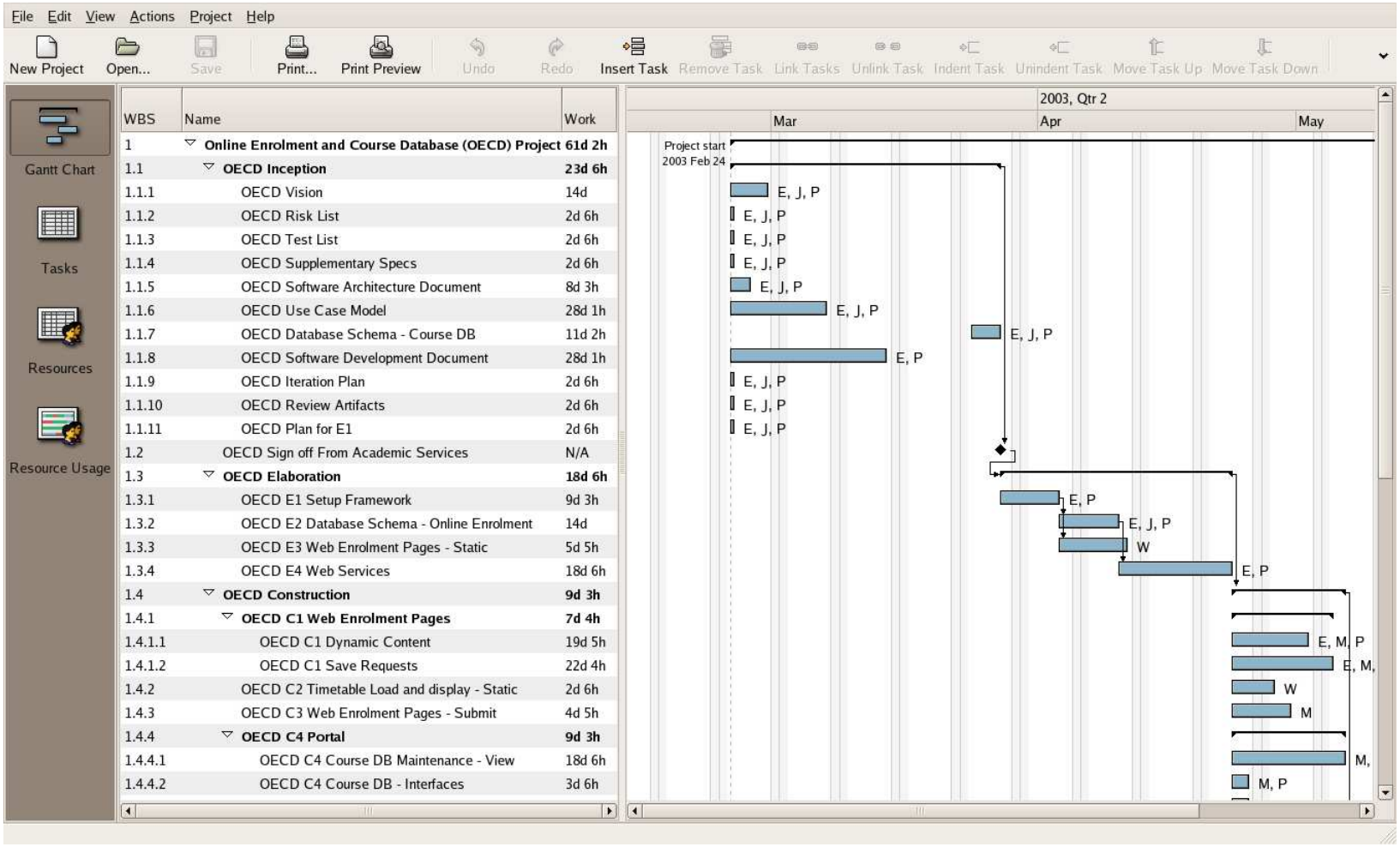
plan lacked many tasks that it should have had, according to our PMM description. Some of the tasks of the human–generated plan were also in the wrong RUP phases, ie: they were out of sequence, so that, for example: the task of capturing supplementary requirements was in the Inception phase of the human–generated plan, when the RUP description we used specified that this task should be in the Elaboration phase.

Resources

The human–generated plan had relatively anonymous resource names, making it difficult to ascertain what kind of resources the project required. The "resource type" aspect of the RUP PMM had been lost from the original project. To remedy this, we produced table 9.1, that relates the original project resources to the PMM ResourceTypes listed in our PMM–created project. We were able to tabulate the PMM ResourceTypes against project resources because we were able to collate similar tasks in both projects, as we described in section 9.1.2. Having done this, we could see that certain sets of resources were assigned to tasks for which our PMM specified particular sets of ResourceTypes, summarized in table 9.1 (c). For example: we can see that each of the project resources M1, M2 and P1 are assigned, with no assistance, to Feature construction, and so are expected to fill the roles of Customer test team, Systems analyst, Data Analyst and Programmer Team, which might be a bit of a stretch for one person. Also, there is a risk created by having programmers test their own code. Ideally, a different person should be assigned to test the code.

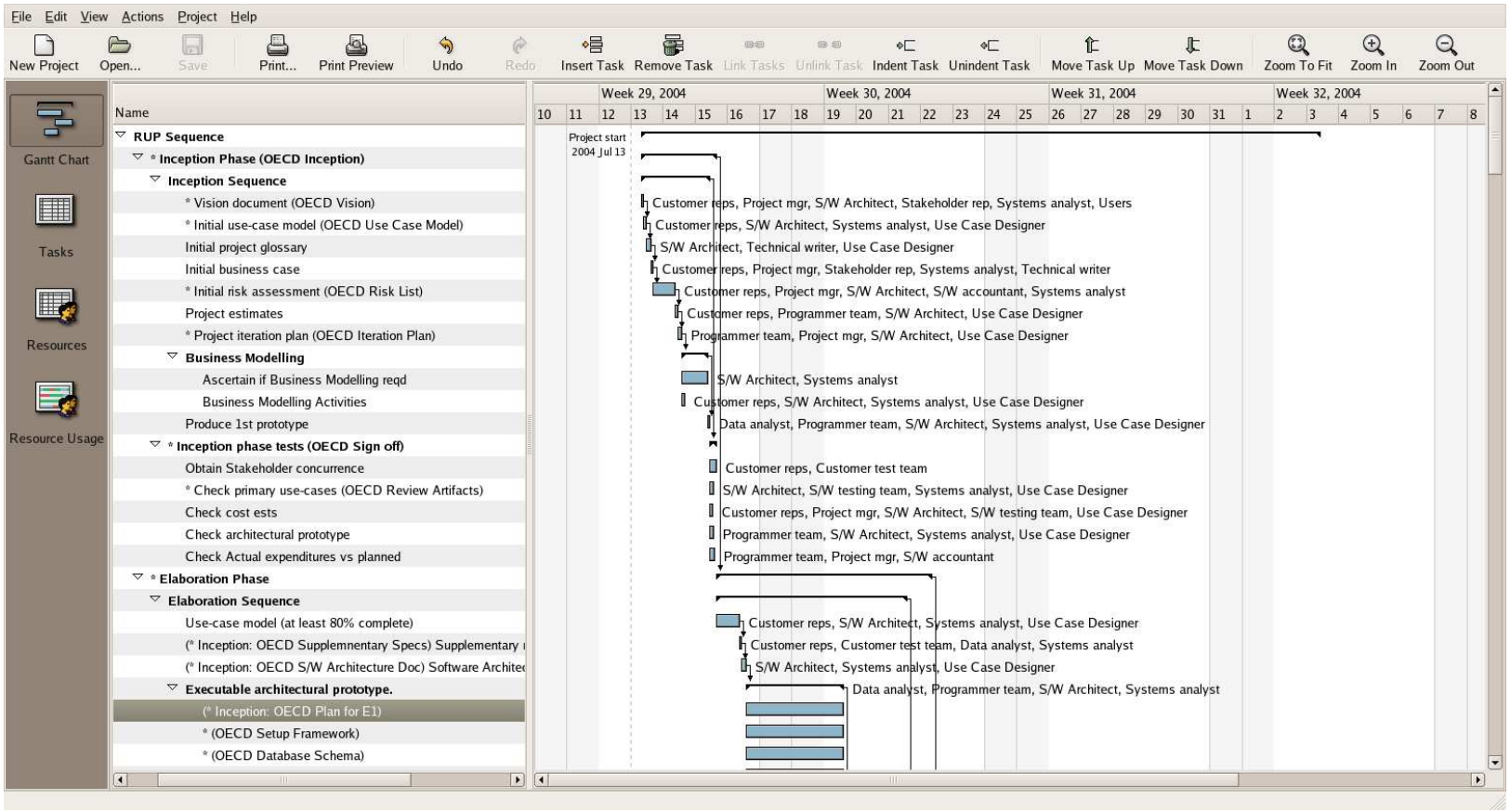
9.1.3 Conclusions

The results of our case study lead us to believe that our framework may assist project managers, by providing tools and visualizations that make the application of PMMs to projects less laborious, and project plans easier to revise. We found obvious correspondence between the tasks of the both the human and PMM–generated plans, so that the four RUP phases of Inception, Elaboration, Construction and Transition existed in both plans, and in the same order. The human–generated plan quickly moved on to implemen-



WBS	Name	Work
1	**Online Enrolment and Course Database (OECD) Project**	**61d 2h**
1.1	**OECD Inception**	**23d 6h**
1.1.1	OECD Vision	14d
1.1.2	OECD Risk List	2d 6h
1.1.3	OECD Test List	2d 6h
1.1.4	OECD Supplementary Specs	2d 6h
1.1.5	OECD Software Architecture Document	8d 3h
1.1.6	OECD Use Case Model	28d 1h
1.1.7	OECD Database Schema - Course DB	11d 2h
1.1.8	OECD Software Development Document	28d 1h
1.1.9	OECD Iteration Plan	2d 6h
1.1.10	OECD Review Artifacts	2d 6h
1.1.11	OECD Plan for E1	2d 6h
1.2	OECD Sign off From Academic Services	N/A
1.3	**OECD Elaboration**	**18d 6h**
1.3.1	OECD E1 Setup Framework	9d 3h
1.3.2	OECD E2 Database Schema - Online Enrolment	14d
1.3.3	OECD E3 Web Enrolment Pages - Static	5d 5h
1.3.4	OECD E4 Web Services	18d 6h
1.4	**OECD Construction**	**9d 3h**
1.4.1	**OECD C1 Web Enrolment Pages**	**7d 4h**
1.4.1.1	OECD C1 Dynamic Content	19d 5h
1.4.1.2	OECD C1 Save Requests	22d 4h
1.4.2	OECD C2 Timetable Load and display - Static	2d 6h
1.4.3	OECD C3 Web Enrolment Pages - Submit	4d 5h
1.4.4	**OECD C4 Portal**	**9d 3h**
1.4.4.1	OECD C4 Course DB Maintenance - View	18d 6h
1.4.4.2	OECD C4 Course DB - Interfaces	3d 6h

(a) the original manually created RUP project plan



(b) the RUP project plan generated by Project Mentor.

Figure 9.1: *Two screen shots of Planner comparing Gantt charts of a manually–generated plan to a plan generated with our Project Mentor tool.*

Project Resource	**Assigned to project tasks**	**equivalent PMM–generated tasks**	**PMM resource types required**
E1, J1, P1	OECD Vision, Risk List, Test List, Supplementary Specs, S/W Architecture document, etc	Vision document, Initial use–case model, Initial risk assessment, Project iteration plan, etc	Users, Stakeholder representative, Customer representatives, Customer test team, Data analyst, Programmer team, Project manager, Software accountant, Software architect, Software testing team, Systems analyst, Use case designer
E1, P1	OECD E1 Setup Framework, E4 Web Services	Executable architectural prototype	Data Analyst, Programmer team, S/W Architect, Systems analyst
E1, M1, P1	OECD Dynamic Content, C1 Save Requests, C6 Attach new Service, C6 Action Requests	Feature construction	Customer test team, Programmer Team, Software Testing Team
M1	C5 Contact Centre—Pack Requests	Feature construction	Customer test team, Systems analyst, Data Analyst, Programmer Team
M2	C7 Change Requests	Feature construction	Customer test team, Systems analyst, Data Analyst, Programmer Team
P1	C8 PCR Integration, Pack Requests	Feature construction	Customer test team, Systems analyst, Data Analyst, Programmer Team
M2, P1	C4 Course DB features, etc	Feature construction	Customer test team, Systems analyst, Data Analyst, Programmer Team
M2, Z1	OECD C9 Addr Table data migration, Transcripts	Feature construction	Customer test team, Programmer Team, Software Testing Team
W1	OECD Web Enrolment pages (static), Timetable load and display (static)	Executable architectural prototype	Systems analyst, Data Analyst, Programmer Team, Software architect

Table 9.1: *Tabulating resources and tasks against the equivalent PMM–generated tasks and resources. This table shows how many resources had to be "Jacks of all trades" in the human–generated project.*

tation, without much planning and designing, missing out and re–ordering PMM steps along the way. This re–ordering of steps is significant because it is unlikely that we have simply misinterpreted the human–generated plan when we classified its tasks. Instead, we speculate that the human project designer has altered the RUP PMM for their own purposes. Apart from the tasks being in a different order, the PMM–generated plan is larger and more prescriptive than the human–generated plan.

Table 9.1 is used to indicate skill sets that particular people or groups of people in the original project are expected to have, according to our PMM. It is then possible to ascertain whether particular groups of people should be retrained or augmented by skills of the specified type.

Our framework provided the context for this case study because it relates PMM data to project data. Because we were able to find traces of PMM information in the human–generated plan, we were able to meaningfully relate the human–generated project data to our PMM–generated project data. For this case study we had to apply a lot of interpretation to the two sets of project data, but our framework affords the possibility of formally testing the conformance of project data to a PMM, using XML schemata for example. Appendix D describes how we used such an approach to schematically verify project data produced by our Project Mentor prototype implementation.

9.2 A Survey to Look For PMM Features in Projects

Our case study of section 9.1 showed that project managers can and do make use of PMMs, and that we can find traces of the application of a PMM in project data. We decided to survey some project data, looking for features that we could relate to PMM concepts. The sample of project data we surveyed was drawn from plans produced using Microsoft Project, and published on the internet[1].

We wanted to test whether PMM ideas were already used by project managers, who might thus benefit from tools and visualizations produced in the context of our framework, that assist them to apply and visualize

[1] a Google search for `Start Finish Duration Work Cost filetype:mpp` produces nearly 10,000 hits at the time of writing

PMMs. To do this, we measured the occurrence of the following features that we relate to PMM ideas:

- A milestone is a task type that tries to act like a PMM control structure, because it specifies a test or sign–off that should complete for the project to proceed. If a milestone is not completed then the project should be revised—more tasks might have to be inserted, or work might have to be discarded. An ordinary project in this situation suffers from the problems that we describe in chapters 2 and 4, but applying a PMM can assist the project manager to revise the project plan, because such a milestone test can be related to the flow of control of a PMM.

- Microsoft Project makes available a "recurring task" facility. This is a task that is set to recur at a set interval for a set number of times, and it is a PMM–like feature, similar to the WhileStep or a RepeatStep that we define in our framework, but lacking the control flow—the task simply recurs for a preset number of times. However, a recurring task introduces the idea of tasks that repeat into projects, and so they relate to the repetition concept of PMMs.

- Tasks with predecessor tasks could relate to a PMM sequence or repetition structure, as we describe in figure 5.5 on page 50.

In addition to the above PMM features, we looked for features that indicated that projects could benefit from the increased automation and descriptive capabilities that our framework affords. In chapter 8 we described how, because of the difficulty of manually recording all the detailed tasks in a project, project managers may "hide" PMM information in project tasks (illustrated in figure 8.1 on page 81).

If the shortest task length of the projects we survey is relatively large—in the order of days rather than hours or minutes—then this may indicate a similar under–recording of project complexity, since the shortest task length in a project indicates roughly the level of detail to which a project plan works. The number of resources in a project may also give an indication of project complexity: we speculated that a large project, over time, would use dozens

of different resources, and a project with only a few resources would either not be expected to do much work, or to have under–represented the number of resources.

9.2.1 Method

We used a sample of 116 Microsoft Project files, gathered from such institutions as city councils, universities and health services—see appendix E for the files. The project files were converted to XML by hand, using Microsoft Project. We created an XSL transform, mspstats.xsl, to gather the following measurements from the XML files:

- Number of tasks
- Shortest Task Length, in minutes
- Number of resources
- Number of predecessors/number of tasks
- Number of milestones/number of tasks
- Number of recurring tasks/number of tasks.

9.2.2 Observations and Discussion

Our observations are summarized in table 9.2. The distributions of our measurements were quite skewed and long–tailed. For this reason, we used the median and upper and lower quartiles for analyzing the measurements, because they are relatively resistant to the statistical outliers that may be present in our measurements. We will discuss each measurement in turn.

Number of tasks The median number of tasks (54), and even the maximum (760), is lower than we expected for recording the real number of tasks in projects. We speculated that the largest projects would consist of thousands of tasks.

Measurement	Lower quartile	Median	Upper quartile
Number of tasks	26	54	126
Shortest Task Length, minutes	0	0	480
Shortest non-zero Task Length, minutes	420	480	480
Number of resources	1	5	16
Number of predecessors/number of tasks	11%	57%	74%
Number of milestones/number of tasks	0.3%	9%	18%
Number of recurring tasks/number of tasks	0%	0%	0%

Table 9.2: *The measurements from our survey of project data.*

Shortest Task Length, minutes The most common shortest task length is zero minutes, but this is because the milestone tasks that most projects have are given a zero duration by Microsoft project. We found that for the vast majority of projects, the shortest non-zero task length was 480 minutes, or one working day. This is the default task length when a new task is created in Microsoft Project 2003, and we found a cluster of 63 measurements on 480 minutes. Only a few projects recorded shorter minimum task lengths, down to 12 minutes.

Number of resources This measurement is also unexpectedly low, with quartiles of one, five and 16. A project plan with one or zero resources probably under–describes a project, and yet 25% of projects fell in this range.

Number of predecessors/number of tasks More than half the tasks in most projects have predecessors, indicating that many project task lists are quite structured.

Number of milestones/number of tasks The average percentage of milestones in a project is 9%, indicating that project managers do make use

of this facility. The percentage of milestone tasks in a project is always likely to be fairly small, since a milestone is the culmination of a series of earlier activities.

Number of recurring tasks/number of tasks Recurring tasks were not used in significant amounts. It may be that this facility is not often useful, or is too complicated to use routinely, or project managers were simply not aware of it.

9.2.3 Conclusions

Our measurements of the number of tasks and resources, and the shortest non-zero task length, indicate that many project plans are not intended to be exhaustive specifications of a project, below the level of tasks one day long. We speculate that the reason for this is that it would be very laborious to record absolutely every task and resource for a project plan, even if the project manager used a template plan. A very detailed plan would in any case have to be constantly and extensively revised, if it were to record every twist and turn of the real progress of a project.

Our framework affords the use of PMMs to automatically create and revise project plans that are much more prescriptive than the manually–created plans our survey found. Tasks could be realistically specified down to the level of hours, for example. Very specific plans, created from very large PMMs, may be useful for micro–managing projects where very detailed project control and tracking is possible and desirable. Large IT projects might be one area for applying such prescriptiveness, because of their poor track record for success, and the amenability of software development technology such as IDEs to instrumentation. On the other hand, a very prescriptive PMM would not suit the high–level style of project planning adopted in the majority of projects we surveyed. For this style of project management PMM designers may want to deliberately keep their PMMs quite simple and be a summary of the PMM, rather than a description of every last detail.

We found that project plans often include some kind of test or control, in the form of milestone tasks. When a milestone fails, a project manager is thrown back on their own skills to revise a project plan, as we showed in

chapter 4. Our framework augments project data with PMM concepts, so that milestones could be translated to PMM control structures, rather than simply being a test that "has to pass". Such PMM–driven projects may thus be less fragile because a PMM specifies the revision of a project, according to its control structures.

Our survey indicated that the PMM–like feature of recurrent tasks added to Microsoft Project is not much used. Our framework allows an approach to augmenting projects with PMMs that may be more acceptable to project managers, since it does not require project tools to have more features added in order to use PMMs. Instead, a PMM can manipulate a project in isolation from the project tool, and so, for example: a project manager does not have to deal directly with a PMM. Where it is necessary for project managers to deal with PMMs and relate them to projects, our framework affords the creation of visualizations to assist them, such as those we described in chapter 8. To summarize our findings from the survey:

- Project plans appear frequently to be more a summary of a project, than an exact description.

- We identified situations where tools operating in the context of our framework could be useful to project managers. For example: we could use a PMM to automate aspects of creating project plans that are more detailed than human–generated plans. Also, project milestones could be translated to PMM control structures that assist the revision of a project plan, rather than a project manager having to manually revise a project plan when a milestone fails.

- Project managers appear not to use advanced features such as recurrent tasks in Microsoft Project. Our framework has the advantage of augmenting projects with PMM concepts, without requiring project tools to be augmented with more features.

9.3 Conclusion

The case study showed that our framework is an effective context for testing the PMM conformance of a project, and revising either the project or the PMM in the light of the conformance test. The survey showed that project managers do make use of PMM–like features such as milestones, and that project managers may benefit from a PMM–driven project that assists them, say: when a project milestone fails. We also found in both the survey and case study that PMM–driven projects can be more complete and descriptive than most manually–created projects. Project managers may benefit from this extra detail in a PMM–driven project, or alternatively PMM designers might have to be careful not to be over–prescriptive when creating PMMs.

Chapter X

Conclusion

In this chapter we summarize our contributions to knowledge. We then recap how we addressed the research problem of chapter 2, and conclude with directions further work could take.

10.1 Our contributions

The main contribution to knowledge of this thesis has been our framework, that consists of a generalized description of the PMM and PM domains, and the connection between them, shown in figure 10.1. The top two quadrants of figure 10.1 are the data models that we derived for PMMs and PM, and these models describe data in the bottom two quadrants. Our framework makes a semantic link between PMMs and PM, by linking PMMs, Steps and Resource Types to Projects, Tasks and Resources.

The generalized, database–level connection between PM and PMM that we have created with our framework affords the creation of tools, metrics and visualizations for PMM and project data, even for project tools originally

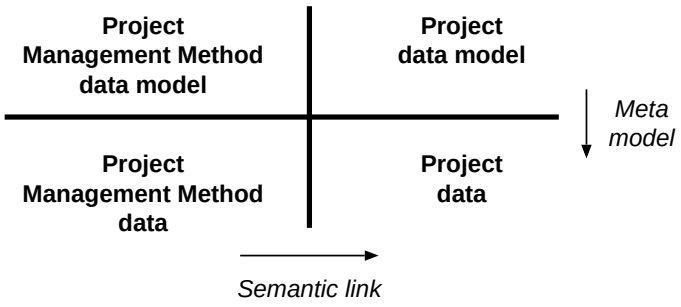


Figure 10.1: *Summary of the concepts in our framework.*

created without reference to our framework, and we have described some prototype examples of these in chapter 8. Many advantages accrue from our framework:

- Our Project Mentor tool applies our framework and can create projects, according to a chosen PMM, in a much less laborious way than manually creating than. This automated approach eases analysis of many different project scenarios, for example.

- Not only is project creation less laborious, it is more repeatable. The most suitable PMM for a particular goal can be selected from a library of PMMs, and consistently applied. The decision support available to a project manager using a suitable PMM may reduce the risk of a project.

- Our framework connects PMMs to projects, and we can use it to create visualizations of PMMs, as they are applied or post–hoc. Project Mentor can make use of Nassi–Shneiderman (NS) diagrams to indicate the operation of the PMM it is applying. We also implemented an XSL transform to create post–hoc visualizations of PMM application in project data, again using NS diagrams.

10.2 Recap

Projects fail, sometimes in expensive and catastrophic ways. The research problem we proposed in chapter 2 was to create a data–level connection between the domains of project management (PM) and project management methods (PMMs), so we can create software tools and visualizations, etc, that use both the reactivity and prescriptiveness of PMMs, and the simplicity, history and context of PM. We believe that these tools can assist project managers to create projects that are more robust, less laborious to manage, and less dependent on highly skilled individuals for success.

By analyzing the domains of PM (in chapter 3) and PMMs (in chapters 4 and 5), we have created data models that describe them both. In chapter 6 we then created a framework that links PM data to PMM data using

well–defined semantics, that we derived by examining the action of PMMs when applied to a project goal. Our framework is general–purpose, and independent of any particular software architecture, as we showed in chapter 7. Implementation–level descriptions of the framework and its semantics are in appendices B and C. In table 2.1 on page 11, Bussler [12] identifies a number of mapping problems from the PMM domain to the PM domain. We formulated solutions to these problems, and table 6.2 on page 63 lists how our framework addresses Bussler's list.

Our framework of figure 10.1 affords the creation of software tools and visualizations that move easily between the two domains, making use of the best features of both. Some example applications of our framework are described in chapter 8. Because our framework is data–based, project data can be directly related to PMMs in a number of ways. Our Project Mentor implementation, for example, stores PMM state directly as project data that represents the serialized PMM process. A PMM can be interrupted and restarted at any time. We created visualizations of PMMs using process visualizations (Nassi–Shneiderman diagrams). Because our framework relates PM data directly to PMM data, it allows us to visualize PMM activity in project data, post–hoc and without reference to any PMM state.

Once we had created data models for projects and PMMs, and a framework that related them together, we were able to look for PMM–related features in a survey of project data, described in chapter 9. We looked for the use of such features as project milestones, because projects augmented with tools that use PMMs might assist the project manager revise a project when, say, a milestone fails to pass.

Our survey also brought to light the summary aspect of many project plans. Because our framework affords the creation of tools that automate project creation and revision, tools produced using our framework may be able to assist project managers with more fine grained management of projects.

Our case study provided some insight as to how a project manager applied a PMM to a realistic situation: the PMM was customized for the project, with steps re–ordered or left out. Our framework allows us to point out the discrepancies between the project data and either say what it "should" be or, alternatively, to indicate how to customize a PMM by, for example:

indicating what steps to leave out or re–order.

10.3 Future Work

10.3.1 Using Commercial Workflow Languages for PMMs

The applications of our framework described in chapter 8 use a PMM description language that was created by a simple transcription of our UML model of PMM data to TPMM, a custom XML schema. This was done partly to test our PMM data model, but also because of a deficiency in the many commercial workflow languages that we might have used to describe PMMs. During our background work for this thesis, we examined some commercial workflow languages such as BPML [8], BPEL4WS [7] and XPDL [94]. We found that these industry standard workflow languages have little or no support for representing resource requirements. Russell, et al [75] find that BPEL4WS has no support at all for resource allocation, for example.

We expect that these commercial workflow languages could express the control flow of PMMs, because workflow is similar to the control flow of PMMs. Provided that some way of describing resource requirements for a workflow language could be found, the language could be used to write PMM descriptions. One way of doing this might be to augment the workflow language with resource descriptions: Workflow activities could have resource requirements added to them. A workflow system that executes a language such as BPEL4WS could then be augmented with resource allocation algorithms. Alternatively, a "resource broker" process might be used to allocate resources to workflow activities, according to their requirements.

Provided the resource description and allocation problems above can be solved, we could create an implementation of our framework that used a commercial workflow language for describing and applying PMMs.

10.3.2 Creating Project Metrics Using Our Framework

Our framework links PMM data to project data, and so we can derive PMM–related metrics for project tasks and resources, because PMMs, steps and resource types used to generate project data are related to each project, task

and resource. If we record project tasks at a fine level of detail; perhaps by instrumenting the tools that are required for the tasks, we could compare very closely the real progress of a project with the planned project, or use such information for deriving project metrics. Even if such a comparison is done post–hoc, it may provide valuable information to a project manager. A library of PMM "best practices" could be built up, to document the most effective ways of applying a PMM, for example.

An example metric could be the number of iterations of a code–test cycle for implementing a software feature. If the iterations exceeded say, five, then something may be wrong with either the specification or the implementation of the software feature, and it should be reviewed.

The Nassi–Shneiderman visualization of project data we described in chapter 8 can assist a project manager to define suitable metrics for a particular PMM. For example: a constantly–cycling sequence of steps would be made obvious by the visualization, as activity for a sequence of tasks would be centred on a single loop in the PMM. Figure 8.6 on page 90 illustrates our visualization highlighting the requirements gathering loop in the Waterfall method. If this same step had been highlighted several times already then there may be cause for concern about the number of iterations of this step.

We can also apply project metrics to PMMs, so that we could, for example measure the relative costs of particular PMMs. The application of a number of PMMs to the same goals could be simulated, as we did in chapter 5, and the resultant project costs measured. This could be done for a number of different project scenarios.

10.3.3 A Hybrid Approach to Workflow Resource Allocation

As we mentioned in section 10.3.1 above, commercial workflow languages do poorly when it comes to dealing with resources. The result of this situation is illustrated in figure 10.2: the relationship between task production and resource allocation is quite loose, and resource allocation may be a completely separate process that follows the production of tasks. Russell, et al[1], describe 43 patterns for how these resource allocation processes work, which

[1] See also `http://is.tm.tue.nl/research/patterns/`

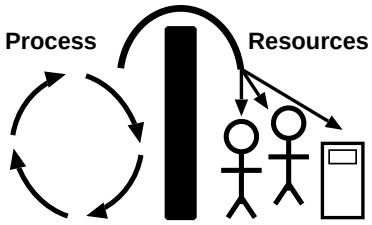


Figure 10.2: *Workflow processes "throw tasks over the fence". Tasks are produced and then handed out to resources for execution.*

are variations on:

- directly allocating resources to tasks.
- allocating resources via roles, or some other resource attribute similar to the `ResourceType` attribute defined in our framework.
- allocation via some kind of resource broker system; tasks might be "auctioned" to resources, for instance.

These patterns are strategies to cope with the stream of tasks produced by a workflow system. Because the allocation of resources to workflow tasks is done as the workflow unfolds, workflow systems produce problems such as:

- Unpredictable behaviour of workflow, and of allocations.
- Overloading of resources.
- Lack of history and context: resources operate in isolation from the workflow.
- Because tasks are "thrown over the fence", it is difficult to re–order or otherwise change tasks to better fit resource availability.

The patterns Russell defines attempt to deal with these kinds of problems, by defining such behaviours as delaying the start of tasks, reallocating tasks to less–loaded resources, etc.

Our Project Mentor implementation of our framework can and does use a "traditional" workflow approach to resource allocation. We allocate resources to tasks depending on a `ResourceType` attribute, or role. However, our framework makes a close link between processes and projects, and so we can also make use of some project–domain strategies for resources allocation, in addition to the process–oriented approach that traditional workflow uses. Hence we can make use of reactive and prescriptive processes and also the simplicity, history and context of projects, when allocating resources in our framework.

When it is applying a PMM, Project Mentor produces "chunks" of project plan (say, by listing all the tasks in a sequence), and so we know what tasks and resources will be required in the near future of the project. To go further into the future, we could also estimate how PMM control steps will go and plan even further ahead. After these "chunks" of project plan are produced and recorded in a project plan, allocation of resources to tasks can be manipulated using PM tools—for example: Planner can change resource allocations in projects produced by Project Mentor. Once we are working in the project domain, we can apply PM resource allocation methods to these "chunks" of project plan.

In the project domain, resources are directly allocated to tasks. However, because a project plan has available to it the project history and future context, project planning has available to it ways of allocating resources that workflow systems do not:

- Workflow systems can delay the start of tasks, but our framework provides the context to delay tasks with reference to critical path requirements of projects, so that we know the latest or earliest possible start for tasks, before progress is impacted.

- Workflow systems can re–order tasks, but our framework provides the context of a project plan, so we know it is OK to re–order unsequenced tasks, for instance.

- We can split tasks (eg: between 2+ resources, or halt/restart a task) in the context of a project plan, and know what the impact will be.

Our framework has available yet another way of dealing with resource allocation problems. Project Mentor can roll back the project plan to some arbitrarily saved state, and begin again. Such an approach is required, even for workflow systems, because some number of workflow tasks might fail to complete. To cope with this situation, BPML and BPEL4WS have the idea of "compensation" activities. However, it is up to the workflow programmer to programme the compensation activity.

The design of Project Mentor not only allows the above "compensation" approach, but we can also take a snapshot or checkpoint of the state of a PMM, by serializing the state to a project instance. Having done this, we can carry on with a re–activated PMM, and roll it forward to plan the near future. If we decide that this particular "chunk" of the project is unsatisfactory, we can re–start the PMM from our snapshot and start again, with different control inputs to the PMM. The advantages we gain from this hybrid approach to resource allocation and project creation include:

- More context for delaying, re–ordering or splitting tasks.
- We know better what we can change in the project plan before impacting on progress.
- We can apply project–domain algorithms eg: for resource levelling.
- We can try and plan the future (by running workflow into the future).
- We can use scenario analysis by trying many different project plans for a goal.

Project Mentor was initially created to bring the power of workflow to project planning. However, we believe that project planning strategies can also benefit workflow, and improve resource allocation and task creation strategies. Ordinary workflow approaches have some difficulties with resource allocation, and the Project Mentor approach may solve some of these problems.

10.4 Concluding Remarks

This thesis has taken a data–modelling approach to try and find ways to improve the control of complicated projects such as IT projects. We propose a framework that links data from projects to project management methods (PMMs), so that projects can make use of the familiarity, history and context of project management (PM) tools, as well as the prescriptiveness and reactivity of PMMs. The data–modelling approach has advantages over other architectures such as message–passing, because our framework is generalized and extensible, and PMM data added to project data is persistent. Project tools can be augmented with PMM concepts, even though they may not have been designed to use them. To demonstrate the possibilities, we have created prototypes of some of the tools and visualizations that our framework affords. We used a case study and a survey to test whether project managers make use of PMM concepts, and, since they do so, tools, metrics and visualizations created in the context of our framework may assist the management of complicated projects.

Appendices

Appendix A

Waterfall and Extreme Programming PMMs

In chapter 5 we applied two PMMs; the Waterfall Method and Extreme Programming to a simple example project. This appendix describes PMMs using text, and using Nassi–Shneiderman diagrams [59].

A.1 The Waterfall Method

First described by Royce [73], the Waterfall Method consists of the following steps:

Requirements gathering. Requirements are determined by the client, and then validated by a software quality assurance team (SQA).

Specifications are produced in the form of documents stating what the program has to do. The requirements are validated by a SQA team and then signed off (agreed to) by the client. A software project management plan is drawn up, with time and cost estimates for the production of the software. This plan is also validated by a SQA team.

Design. The specifications are used to produce the design for the software—how the software will work rather than what it will do. Acceptance tests for the software are also produced. In the light of the design activities, faults with the specifications may become apparent, requiring revisions to both design and specifications. These changes to specification must be documented and signed off by the customer and the SQA team.

Implementation. This step constitutes the production of code, or the realization of the design. Once again, the implementation step may bring

to light faults in the specification and design of the program, resulting in changes to the specification, design and implementation, similar to changes produced during the specification step. These changes must also be signed off by the customer.

Once the program is completed, SQA team and the client sign off the program according to the agreed acceptance tests.

Integration and deployment. The completed software is installed on the client's computer systems and integrated into the business organization. The client must agree and sign off that the installation is complete—this agreement can be difficult to obtain!

Maintenance. Once the completed system is installed, any further changes to the system constitute maintenance. Eventually, the maintenance may be substantial enough to require a complete new software project.

In the above description, and as originally envisaged by Royce, failure of any validation and verification step may require the Waterfall method to restart from an earlier step. Figure A.1 illustrates our interpretation of the Waterfall process[1].

A.2 Extreme Programming

Extreme Programming (XP) [5][46] is a modern software engineering method that is extremely iterative, and aims to "release early and often" updates to the software being developed. XP comes with a number of rules and practices:

Schedule: The schedule of the project is closely managed, and normally there are regular releases of software according to a schedule of a few days up to a fortnight.

Test–Driven Programming: Before any code is written to implement a user requirement, a test for the code (unit test) and test for the customer

[1] This Nassi–Shneiderman Diagram was produced using the "nassi" LaTeXstyle, cf figure A.2 that was produced directly from our SVG visualization of a PMM

PMM — Waterfall

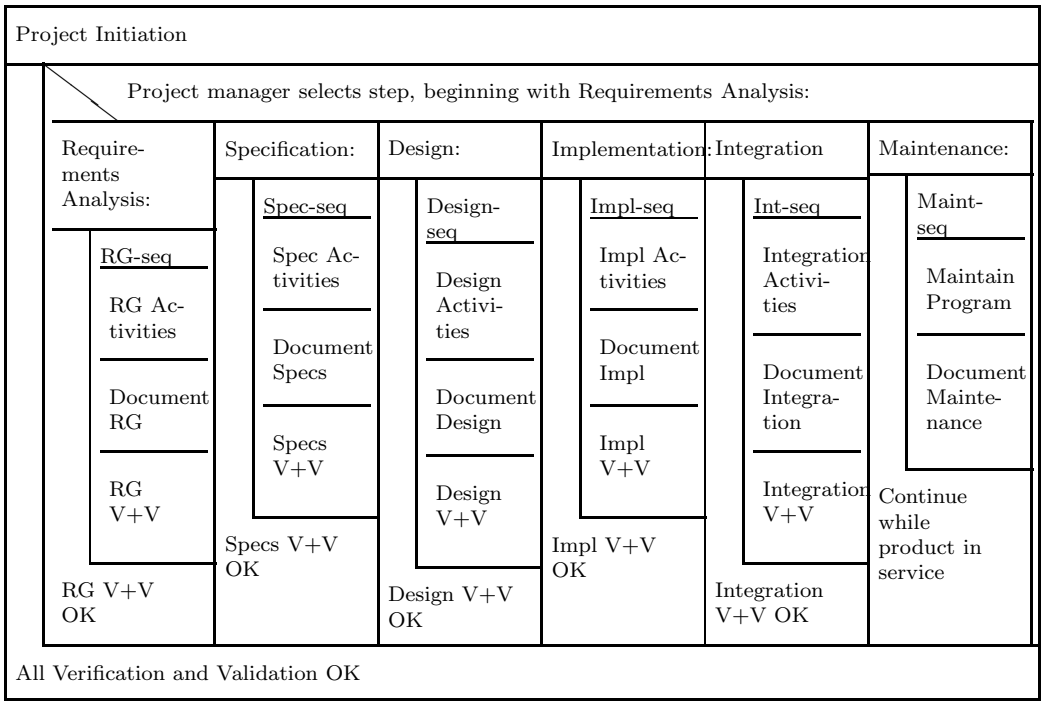


Figure A.1: *The control flow of the Waterfall PMM.*

(acceptance test) is written.

Pair Programming: Programmers work in pairs on the same task, providing a constant review of the code.

Refactoring: As a result of the constant review of the code, refactoring (rewriting the software) is a frequent activity, on the grounds that it is good to frequently rewrite code, and rewriting will simplify and improve the code. For example, one goal for object–oriented projects would be to increase cohesion and reduce coupling of software modules.

The code must always build: It is a requirement that each incremental change to the project produces a usable and running system, so that the all code is always integrated.

The XP method applies the above rules and practices to the following steps:

System Architecture The system architecture, around which the initial code will be written, is designed. The architecture provides a starting point for the implementation of the first release, then the following sequence is iterated to produce software releases until the customer decides to finish the project:

Release Planning This is carried as follows:

- User requirements are written down as "user stories", typically on a small card, and these provide the specifications for the coding. The client decides which stories have the highest priority for implementation.
- User stories are broken down into tasks of a few hours length.
- The programming team estimates the length and difficulty of the tasks, and the customer decides which stories are most important for this release of the software.

Implementation: The following steps iterate frequently within a release, and may take place in parallel as tasks are implemented.

- A "design spike" may be required, where the whole implementation team discusses a problematic code feature. The result may be the subdivision of the feature, or it may be moved down in priority for implementation.
- The first action of an implementation task is the creation of a test case (unit test) for the code implementing the task. Tests are created with consultation between the programmer and the customer.
- The code is implemented and tested with the unit test, which has to run correctly to pass.
- The customer decides the acceptance, or not, of implemented features—they may respecify a feature once they see it in operation.
- A feature is not implemented successfully until it is integrated into the current software release, and all unit tests for the release run.

Release: Features that have been implemented are released to the customer, strictly according to the preset release schedule.

Termination: The project may be stopped after any release, because the software must pass all tests to be released.

Figure A.2 illustrates our interpretation of the Extreme Programming PMM that we created for Project Mentor. The diagram was produced using batik [4] to render an SVG graphic produced using pmmtosvg.xsl [23]. Pmmtosvg.xsl was applied to the XML file containing our description of the Extreme Programming PMM.

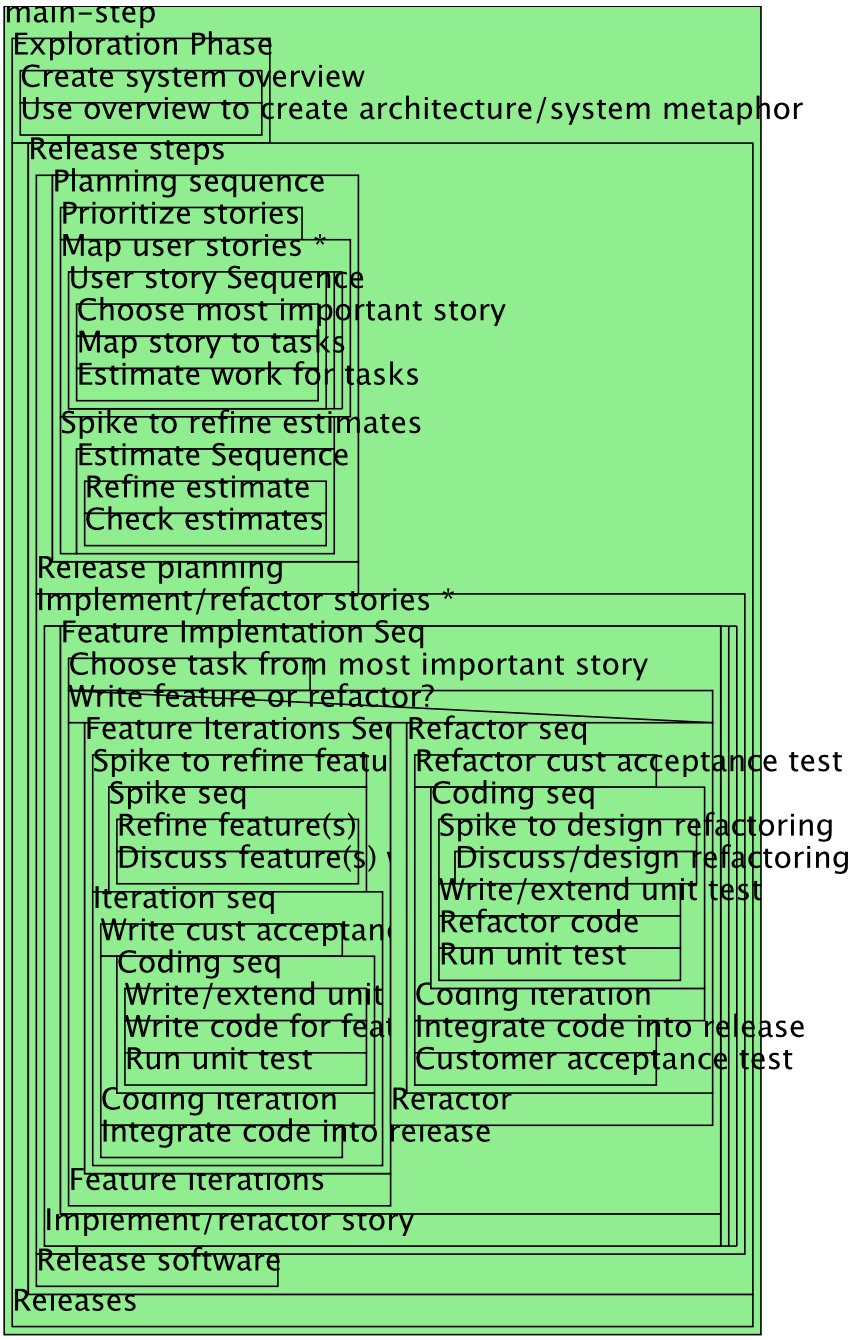


Figure A.2: *Nassi–Shneiderman Diagram of the Extreme Programming process.*

Appendix B

OO Methods and Procedural Code

In chapter 6 we formulated semantics for our framework, linking PMM to PM data. These semantics were described at the conceptual level, and they can be fleshed out by considering an implementation using a specific architecture. Because the concepts of our framework were described with object–oriented UML, we will begin with an object–oriented implementation description.

B.1 Object–Oriented Methods

We can take the Interpreter pattern that we used in chapter 7 and write down object–oriented pseudocode for object initializers and the `myBehaviour()` methods that we described. First we describe the PMM methods so created, and then section B.2 describes the PM methods.

PMM.applyToGoal(aGoal: text)

Parameters

`aGoal` is a description of a project Goal to achieve.

Body

```
new PMMproject(aGoal, this)

parentTask = Task.getTask(this.mainStep, NULL, "UNSEQ", NULL)

this.mainStep.myBehaviour(parentTask)
```

SimpleStep.myBehaviour(pTask: Task)

Parameters

`pTask` is ignored, but required for the polymorphic myBehaviour() method.

Body

For each `Requirement` associated with `this` `Requirement.allocResource(pTask)`

SequencedStep.myBehaviour(pTask: Task)

Parameters

`pTask` is the *Task* that will be the parent of all *Task*s created here.

Body

`predTask` = NULL

For each *Step* in `this.contains`

```
predTask = Task.getTask(Step, pTask, Step.seqKind, predTask)
Step.myBehaviour(predTask)
```

UnSequencedStep.myBehaviour(pTask: Task)

We are able to implement concurrent behaviour as threads (lightweight processes) for two reasons. Firstly, there is no communication between each thread. Secondly, each new thread produces a new branch in our tree of project data. Third, there are no user variables, and only the control flow of the immutable PMM is used to drive the process.

Parameters

`pTask` is the *Task* that will be the parent of all *Task*s created here.

Body

for `this.cardinalityExpr` new Thread

```
parentTask = Task.getTask(this.contains, pTask, "UNSEQ", NULL)
Step.myBehaviour(parentTask)
```

Thread.join()

RepeatStep.myBehaviour(pTask: Task)

Parameters

`pTask` is the *Task* that will be the parent of all *Task*s created here.

Body

`parentTask` = NULL

repeat

```
parentTask = Task.getTask(this.contains, pTask, "FS", parentTask)
this.contains.myBehaviour(parentTask)
```

until (`this.ctrlExpr` == FALSE)

WhileStep.myBehaviour(pTask: Task)

Parameters

`pTask` is the *Task* that will be the parent of all *Task*s created here.

Body

`parentTask` = NULL

while (evaluating `this.ctrlExpr` == TRUE)

```
parentTask = Task.getTask(this.contains, pTask, "FS", parentTask)
this.contains.myBehaviour(parentTask)
```

SelectionStep.myBehaviour(pTask: Task)

Parameters

`pTask` is the *Task* that will be the parent of all *Task*s created here.

Body

`selectedStep` = (the result of evaluating `this.mapExpr`)

`parentTask` = `Task.getTask(selectedStep, pTask, "UNSEQ", NULL)`

`selectedStep.myBehaviour(parentTask)`

Requirement.allocResource(aTask: SimpleTask)

Implements a simple resource-allocation mechanism: allocate to the given `SimpleTask` *the first* `Resource` *we find with the correct* `resourceType`*. If none found then create an appropriate* `Resource`*.*

Parameters

`aTask` is the `SimpleTask` for which we will be allocating `Resource`s.

Body

For each `Requirement` found by evaluating `this.allocExpr`

> `aResource` = the first `Resource` found where (`Resource.ResourceType` == `aResourceType`)
>
> If (`aResource` == NULL) aResource = new `Resource(aResourceType)`
>
> new `Allocation(aResource, aTask)`

B.2 PM Methods

In the following sections, we describe initializers for project data created using the "PMM project" model formulated in section 7.1.2:

PMMproject(aGoal: text, aPMM: PMM)

This is the initializer for a PMMproject.

Parameters

`aGoal` is a description of a project goal to achieve.

`aPMM` is the PMM that will be applied to the project goal.

Body

`goal` = `aGoal`

`defaultCal` = new `Calendar("9am to 12pm and 1pm to 5pm, Monday to Friday")` *This will be the default calendar.*

`name` = a name for the `PMMproject`.

`startDate` = the current time

`whatPMM` = `aPMM`

Task.getTask(aStep: Step, pTask: Task, pType: text, predTask: Task)

Returns a SimpleTask or SummaryTask, as appropriate.

Parameters

`aStep` is a PMM *Step*.

`pTask` is a parent *Task*.

`pType` is the type of predecessor.

`predTask` is a predecessor *Task*, or NULL for no predecessor.

Body

if (aStep is SimpleTask) return new SimpleTask(aStep: Step, pTask: Task, pType: text, predTask: Task);

else return new SummaryTask(aStep: Step, pTask: Task, pType: text, predTask: Task);

SimpleTask(aStep: Step, pTask: Task, pType: text, predTask: Task)

This is the initializer for a SimpleTask

Parameters

`aStep` is a PMM *Step*.

`pTask` is a parent *Task*.

`pType` is the type of predecessor.

`predTask` is a predecessor *Task*, or NULL for no predecessor.

Body

`name` = `aStep.name`

`descr` = `aStep.descr`

`milestone` = `aStep.milestone`

`taskType` = `aStep`

`percentComplete` = 0

`startTime` = *calculate from predecessor startTimes and durations.*

`duration` = *a default of one day.*

`parent` = `pTask`

if (`predTask` != NULL) `myPred` = new `Predecessor(predTask, pType)`

SummaryTask(aStep: Step, pTask: Task, pType: text, predTask: Task)

This is the initializer for a `SummaryTask`

Parameters

`aStep` is a PMM *Step*.

`pTask` is a parent *Task*, or NULL for the top-level *Task*

`pType` is the type of predecessor

`predTask` is a predecessor *Task*, or NULL for no predecessor

Body

`name` = `aStep.name`

`descr` = `aStep.descr`

`taskType` = `aStep`

`startTime` = *calculate from predecessor startTimes and durations.*

`duration` = *the the sum of the* Task *durations contained by this task.*

`parent` = `pTask`

if (`predTask` != NULL) `myPred` = new `Predecessor(predTask, pType)`

Predecessor(predTask: Task, pType: text)

This is the initializer for a Predecessor.

Parameters

`predTask` is a predecessor *Task*, or NULL for no predecessor.

`pType` is a string from the set "UNSEQ", "FS", "SS", "SF", "FF". *This is a specifier for the sequence type of the* `predecessor` *for the* Task *created: either no sequence, finish-to-start, start-to-start, start-to-finish, finish-to-finish.*

Body

`pTask` = `predTask`

`pType` = `pType`

`lag` = 0 *default of zero lag.*

Resource(rType: ResourceType)

This is the initializer for a Resource.

`name` = `rType.name`

`calID` = *the default Calendar.*

`myType` = `rType`

Allocation(aRes: Resource, aTask: Task)

This is the initializer for an Allocation.

`thisTask` = `aTask`

`usesA` = `aRes`

Calendar(listOfTimes: text)

This is the initializer for a Calendar.

```
availability = listOfTimes
```

B.3 Procedural Pseudocode

In section 7.2 we describe how to implement our framework using relational tables. These tables require procedural code rather than the object–oriented methods described above. Here is a description of how to apply PMMs described using relational tables:

1. **Create the Project Instance**

   (a) Create a set of empty tables similar to those of table 7.2.

   (b) Create a new row in the `Project` table, and set the fields in the row as follows:

   `id` set to a unique identifier generated for the project.

   `name` set this to a suitable name for the project.

   `descr` set this to a suitable description for the project.

   `start date` set this to the current time, unless the project is to begin at some other, specified, time.

   `whatPMM` set to the `name` of the PMM used for the project, from the PMM table.

   `goal` set to the specified goal for the project.

   (c) In the `Calendar` table, record a row with the the `id` field set to a unique identifier. Set the `availability` field to 9am to 12pm and 1pm to 5pm, Monday to Friday. This will be the default calendar for the project.

2. **Begin applying the PMM** starting with the PMM *Step* identified by the `mainStep` field in the `PMM` table and record a a `SimpleTask` table row, if it is a `SimpleStep`, otherwise record it as a `SummaryTask` table

row. Set the `Project mainTask` field to the id of this first task. The child *Steps* of this *Step* are then applied, then their children, and so on, in a depth–first walk of the PMM *Steps*.

3. **Record Tasks, Resources and Allocations** in the project instance.

   - Whenever a row is recorded in the project `SummaryTask` table, fields in the row recorded are set in this way:

     `id` is a unique identifier generated for each task.

     `name` is set from the `name` of the associated PMM *Step*.

     `descr` is set to an empty string.

     `startTime` is set to the start time of the project, plus the the sum of the task durations which precede this task.

     `duration` is set to the sum of the durations of the tasks summarized.

     `parentID` set to the parent `taskID`.

     `taskType` is set from the PMM *Step* identifier; `id`.

   - Whenever a row is recorded in the project `SimpleTask` table, fields in the row recorded are set in this way:

     `id` is a unique identifier generated for each task.

     `name` is set from the `name` of the associated PMM *Step*.

     `descr` is set to an empty string.

     `startTime` is set to the start time of the project, plus the the sum of the task durations which precede this task.

     `duration` is set to a default estimate of one day.

     `parentID` set to the parent `taskID`.

     `percentComplete` is set to zero.

     `milestone` is set from the Step `milestone` field.

     `taskType` is set from the PMM *Step* identifier; `id`.

   - The semantics of project task creation is specified by the name of table in which each PMM *Step* is found, as follows:

`SimpleStep`

(a) Record a row in the `SimpleTask table`.

(b) i. Search the PMM `Requirements` table for rows having the `id` of this `SimpleStep`.

ii. For each row found, search the project `Resource` table for resources having the same `resourceType` attribute.

iii. If such a resource is not found, then record a new row in the project `Resource` table. Set the attributes of this row as follows:

`id` set to a unique identifier for the resource.

`name` set initially to the PMM `resourceType name` field.

`descr` set initially to an empty string.

`calendarID` set initially to the `Project defaultCal`.

`resourceType` set to the `id` of the `resourceType`.

iv. The association of task to resource is then recorded in the `thisTask` and `usesA` attributes of a row in the `Allocation` table of the project instance.

`SequencedStep`

(a) Record the `SequencedStep` as a row in the project `SummaryTask` table.

(b) Record each child PMM *Step* contained by the `SequencedStep` as a `SimpleTask` table row, if it is a `SimpleStep`, otherwise record it as a `SummaryTask` table row.

(c) Record rows in the `Predecessor` table to record the sequence of steps that is listed in the `ordering` field of the `SequencedStep`. Set the the `predecessor taskID` to the ID of the predecessor task of each task `id`. Set the `pType` from the `seqKind` attribute of the PMM `SequencedStep` to describe the kind of step sequence; finish–to–start, finish–to–finish, start–to–finish or start–to–start (FS/FF/SF/SS).

`UnSequencedStep`

(a) Record the `UnSequencedStep` as a row in the project `SummaryTask` table.

(b) For a number of times set by the `cardinalityExpr` expression, record the PMM *Step* child of this `UnSequencedStep` as a `SimpleTask` table row, if it is a `SimpleStep`, otherwise record it as a `SummaryTask` table row.

`RepeatStep`

(a) Record the `RepeatStep` as a row in the project `SummaryTask` table.

(b) Record the PMM *Step* child of this `RepeatStep` as a `SimpleTask` table row, if it is a `SimpleStep`, otherwise record it as a `SummaryTask` table row.

(c) If this is not the first iteration of the loop, record a row in the project `Predecessor` table with the `predecessor taskID` set to the ID of the task generated in the previous loop, and the `taskID` set to the ID of the task generated in this loop. Set the `predecessor type` to finish–to–start and the `lag` to zero.

(d) If the result of the `RepeatStep control expression` is "failure", repeat this sequence beginning from step 3b, otherwise control flow of this `RepeatStep` ends.

`WhileStep`

(a) Record the `WhileStep` as a row in the project `SummaryTask` table.

(b) If the result of the `WhileStep control expression` is "failure", then control flow of this `WhileStep` ends.

(c) Record the PMM *Step* child of this `WhileStep` as a `SimpleTask` table row, if it is a `SimpleStep`, otherwise record it as a `SummaryTask` table row.

(d) If this is not the first iteration of the loop, record a row in the project `Predecessor` table with the `predecessor taskID` set to the ID of the task generated in the previous loop, and the

`taskID` set to the ID of the task generated in this loop. Set the `predecessor type` to finish–to–start and the `lag` to zero.

(e) repeat this sequence beginning from step 3b.

`SelectionStep`

(a) Record the `SelectionStep` as a row in the project `SummaryTask` table.

(b) Use the `mapExpr` to identify a child *Step* that is then recorded as as a `SimpleTask` table row, if it is a `SimpleStep`, otherwise it is recorded as a `SummaryTask` table row.

Appendix C

XML Schemata used by Project Mentor

This appendix lists the XML schemata used by Project Mentor.

C.1 TPMM Schema

TPMM ("Tony's PMM schema") is the schema we created to describe PMMs, derived from the UML description of PMMs on the left hand side of our framework.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.cosc.canterbury.ac.nz/tony.dale/schemata/TPMM.xsd"
xmlns:tpmm="http://www.cosc.canterbury.ac.nz/tony.dale/schemata/TPMM.xsd"
targetNamespace="http://www.cosc.canterbury.ac.nz/tony.dale/schemata/TPMM.xsd">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      Tony's Project Management Method (TPMM) schema
    </xsd:documentation>
  </xsd:annotation>
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      This schema will validate the following
      stand-alone objects: PMM (Project Management Method)
    </xsd:documentation>
  </xsd:annotation>
<!-- == Global Elements ==================================== -->
  <xsd:element name="PMM">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">This is the root PMM document.
```

```
        All following objects are contained in a PMM.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
    <xsd:annotation>
      <xsd:documentation xml:lang="en">
        A PMM contains one and only one main step, then a collection
        of ResourceTypes, and name and description attributes.
      </xsd:documentation>
    </xsd:annotation>
      <xsd:sequence>
        <xsd:group ref="tpmm:stepKinds" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="ResourceTypes" type="tpmm:typeResourceTypes"
          minOccurs="0" maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:string" use="required"/>
      <xsd:attribute name="descr" type="xsd:string" use="optional"/>
    </xsd:complexType>
  </xsd:element>
<!-- == Complex types ====================================== -->
  <xsd:group name="stepKinds">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">
        This is a group that is shorthand for all the different kinds
        of Step elements we can list:
      </xsd:documentation>
    </xsd:annotation>
    <xsd:choice>
      <xsd:element name="SimpleStep" type="tpmm:typeSimpleStep"/>
      <xsd:element name="SequencedStep" type="tpmm:typeSequencedStep"/>
      <xsd:element name="UnSequencedStep" type="tpmm:typeUnSequencedStep"/>
      <xsd:element name="SelectionStep" type="tpmm:typeSelectionStep"/>
      <xsd:element name="RepeatStep" type="tpmm:typeRepeatStep"/>
      <xsd:element name="WhileStep" type="tpmm:typeWhileStep"/>
    </xsd:choice>
  </xsd:group>
```

```
<xsd:complexType name="typeBaseStep" abstract="true">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      A basic PMM Step from which we derive all other Steps:
    </xsd:documentation>
  </xsd:annotation>
  <xsd:attribute name="name" type="xsd:string" use="required"/>
  <xsd:attribute name="id" type="xsd:long" use="required"/>
  <xsd:attribute name="descr" type="xsd:string" use="optional"/>
</xsd:complexType>
<xsd:complexType name="typeSimpleStep">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      A PMM SimpleStep:
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="tpmm:typeBaseStep">
      <xsd:annotation>
        <xsd:documentation xml:lang="en">
          A SimpleStep may need Resources, so we may have a
          Requirements element containing one more more
          Requirement elements, and an attribute to indicate
          whether PMM Tasks derived from this SimpleStep
          should be milestone Tasks.
        </xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element name="Requirements" type="tpmm:typeRequirements"
        minOccurs="0" maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attribute name="milestone" type="xsd:boolean"
        use="optional"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

```
<xsd:complexType name="typeCompositeStep" abstract="true">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
       A CompositeStep may contain one or more Steps:
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="tpmm:typeBaseStep">
      <xsd:group ref="tpmm:stepKinds" minOccurs="1"
        maxOccurs="unbounded"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="typeUnaryStep" abstract="true">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
       A UnaryStep is a CompositeStep that contains only one Step:
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:restriction base="tpmm:typeCompositeStep">
      <xsd:group ref="tpmm:stepKinds" minOccurs="1" maxOccurs="1"/>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="typeRepeatStep">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      A RepeatStep loops one Step with a test (ctrlExpr) at the bottom:
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="tpmm:typeUnaryStep">
      <xsd:attribute name="ctrlExpr" type="xsd:string" use="required"/>
    </xsd:extension>
  </xsd:complexContent>
```

```
</xsd:complexType>
<xsd:complexType name="typeWhileStep">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      A WhileStep loops one Step with a test (ctrlExpr) at the top:
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="tpmm:typeUnaryStep">
      <xsd:attribute name="ctrlExpr" type="xsd:string" use="required"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="typeSelectionStep">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      A SelectionStep uses the mapExpr to select zero or one Step
      from the set of Steps contained:
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="tpmm:typeCompositeStep">
      <xsd:attribute name="mapExpr" type="xsd:string" use="required"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="typeSequencedStep">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      The steps in a SequencedStep are instantiated to project Tasks
      with predecessors of type seqKind, in XML document order:
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="tpmm:typeCompositeStep">
      <xsd:attribute name="seqKind" type="tpmm:enumSeqType"
```

```
        use="required"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="typeUnSequencedStep">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      The step in an UnSequencedStep is instantiated to N project Tasks,
      where N is a number obtained from the cardinalityExpr.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="tpmm:typeUnaryStep">
      <xsd:attribute name="cardinalityExpr" type="xsd:string"
        use="required"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="typeRequirements">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      The Requirements element is contained in a Step element, and
      contains zero or more Requirement elements:
    </xsd:documentation>
  </xsd:annotation>
  <xsd:sequence minOccurs="0">
    <xsd:element maxOccurs="unbounded" name="Requirement"
      type="tpmm:typeRequirement"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="typeRequirement">
  <xsd:annotation xml:lang="en">
    <xsd:documentation>
      A Requirement has an allocExpr that gives us the ID number
      of required ResourceTypes.
    </xsd:documentation>
```

```
    </xsd:annotation>
    <xsd:attribute name="allocExpr" type="xsd:string" use="required"/>
  </xsd:complexType>
  <xsd:complexType name="typeResourceTypes">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">
        The ResourceTypes element is contained in the root PMM element,
        and contains zero or more ResourceType elements.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:sequence minOccurs="0">
      <xsd:element maxOccurs="unbounded" name="ResourceType"
        type="tpmm:typeResourceType"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="typeResourceType">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">
        The ResourceType element specifies project Resources that
        are required by SimpleTasks:
      </xsd:documentation>
    </xsd:annotation>
    <xsd:attribute name="id" type="xsd:long" use="required"/>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="descr" type="xsd:string" use="optional"/>
  </xsd:complexType>
<!-- == Enumerations ======================================= -->
  <xsd:simpleType name="enumSeqType">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">
        These are the types of sequence allowed in a SequencedStep:
        FF: finish-to-finish
        FS: finish-to-start
        SS: start-to-start
        SF: start-to-finish
      </xsd:documentation>
```

```
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="FF"/>
      <xsd:enumeration value="FS"/>
      <xsd:enumeration value="SS"/>
      <xsd:enumeration value="SF"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

C.2 Using the Planner DTD for PMM–driven Projects

Similarly to TPMM, we could have derived a custom XML schema for project data from our UML model. However, as we noted in chapter 3, our PM model defines concepts found in many existing PM tools. We decided to use the XML schema from the Planner project tool [42] for our Project Mentor implementation, because such an approach allows us to use Planner to view and manipulate project data, saving some implementation effort.

Planner, like many other project tools, provides custom properties that can be assigned to projects, tasks and resources. We have used these properties to augment the Planner DTD with the PMM–to–PM linking attributes required by our framework. Project Mentor, our implementation of the framework, uses a minimal (Planner) project instance to begin applying a PMM. The minimal project ("emptyproj.xml") is a nearly empty project that contains a default project calendar, and also defines the following custom properties:

Project Properties:

PMM The URI of the PMM we used for the project.

Goal Textual description of the project goal.

Task Properties:

stepID The ID of the PMM step used to create the task.

priority This is actually an XML attribute of the Task element, that we have used instead of a custom property. Project Mentor sets the task priority to 9999 to identify tasks that are active, for the purpose of recording the active PMM state in a Planner project. Active tasks are associated with active PMM steps, and so when the PMM is reapplied to the project by Project Mentor, the active state of the PMM can be reconstructed.

Resource Properties:

resourceType The ID of the PMM ResourceType used to create this Resource.

Planner DTD The Planner DTD version 0.6 distributed with Planner version 0.13 doesn't describe all the XML data that the Planner software actually writes: the `Task` element definition in the DTD is missing the attributes `work-start` and `priority`, and the `Resource` element is missing the `short-name` attribute. We have augmented the Planner schema, listed below, with these attributes.

```
<?xml version ='1.0' encoding='UTF-8'?>

<!--
  This is the XML DTD used by Planner version 0.13 with the attributes
  left out by the distributed Planner DTD.
-->

<!ELEMENT project (properties*,phases?,calendars?,tasks?,
resource-groups?,resources?,allocations?)>
<!ATTLIST project mrproject-version CDATA #REQUIRED
                  name              CDATA #REQUIRED
                  company           CDATA #IMPLIED
                  manager           CDATA #IMPLIED
                  project-start     CDATA #REQUIRED
  calendar          CDATA #IMPLIED
  phase             CDATA #IMPLIED>
```

```
<!ELEMENT properties (property*)>

<!--
Note: A mistake in the format design led to this suboptimality. A
      "property" tag can be both a property declaration and a property
      value. When it's a declaration, all attributes except "value"
      are required and allowed, and only "name" and "value" are required
      and allowed. We'll fix this for the new GSF based xml format.
-->

<!ELEMENT property (list-item*)>
<!ATTLIST property name             CDATA #REQUIRED
                  type
              (date|duration|float|int|text|text-list|cost) #IMPLIED
                  owner            (project|task|resource) #IMPLIED
                  label            CDATA #IMPLIED
                  description      CDATA #IMPLIED
                  value            CDATA #IMPLIED>

<!ELEMENT list-item  EMPTY>
<!ATTLIST list-item  value          CDATA #REQUIRED>

<!ELEMENT phases (phase*)>

<!ELEMENT phase EMPTY>
<!ATTLIST phase name                CDATA #REQUIRED>

<!ELEMENT predecessors (predecessor*)>

<!ELEMENT constraint EMPTY>
<!ATTLIST constraint type           CDATA #REQUIRED
                     time           CDATA #REQUIRED>

<!ELEMENT predecessor EMPTY>
<!ATTLIST predecessor id               CDATA #REQUIRED
```

```
                    predecessor-id   CDATA #REQUIRED
                    type             (FS|FF|SS|SF) "FS"
                    lag              CDATA #IMPLIED>

<!ELEMENT tasks (task*)>

<!ELEMENT task (properties?,constraint?,predecessors?,task*)>
<!ATTLIST task id                   CDATA #REQUIRED
               name                 CDATA #REQUIRED
               note                 CDATA #IMPLIED
               effort               CDATA #IMPLIED
               start                CDATA #REQUIRED
               end                  CDATA #REQUIRED
       work-start           CDATA #IMPLIED
               duration             CDATA #IMPLIED
               work                 CDATA #IMPLIED
               percent-complete     CDATA #IMPLIED
               priority             CDATA #IMPLIED
               type                 (normal|milestone) "normal"
               scheduling           (fixed-work|
                                     fixed-duration) "fixed-work">

<!ELEMENT resource-groups (group*)>
<!ATTLIST resource-groups default_group CDATA #IMPLIED>

<!ELEMENT group EMPTY>
<!ATTLIST group id          CDATA #REQUIRED
                name        CDATA #REQUIRED
                admin-name  CDATA #IMPLIED
                admin-email CDATA #IMPLIED
                admin-phone CDATA #IMPLIED>

<!ELEMENT resources (resource*)>

<!ELEMENT resource (properties?)>
<!ATTLIST resource id                CDATA #REQUIRED
```

```
                    name                CDATA #REQUIRED
                    short-name          CDATA #IMPLIED
                    email               CDATA #IMPLIED
                    type                (1|2) #REQUIRED
                    group               CDATA #IMPLIED
                    units               CDATA #REQUIRED
                    note                CDATA #IMPLIED
                    std-rate            CDATA #IMPLIED
                    ovt-rate            CDATA #IMPLIED
                    calendar            CDATA #IMPLIED>

<!ELEMENT allocations (allocation*)>

<!ELEMENT allocation EMPTY>
<!ATTLIST allocation task-id          CDATA #REQUIRED
                     resource-id      CDATA #REQUIRED
                     units            CDATA #IMPLIED>

<!ELEMENT calendars (day-types,calendar*)>

<!ELEMENT day-types (day-type*)>
<!ELEMENT day-type (interval*)>
<!ATTLIST day-type id                 CDATA #REQUIRED
                   name               CDATA #REQUIRED
                   description        CDATA #REQUIRED>

<!ELEMENT interval EMPTY>
<!ATTLIST interval start              CDATA #REQUIRED
                   end                CDATA #REQUIRED>

<!ELEMENT calendar (default-week,overridden-day-types?,days?,calendar*)>
<!ATTLIST calendar     name           CDATA #REQUIRED
       id             CDATA #REQUIRED>

<!ELEMENT default-week EMPTY>
<!ATTLIST default-week mon            CDATA #IMPLIED
```

```
                        tue             CDATA #IMPLIED
                        wed             CDATA #IMPLIED
                        thu             CDATA #IMPLIED
                        fri             CDATA #IMPLIED
                        sat             CDATA #IMPLIED
                        sun             CDATA #IMPLIED>

<!ELEMENT overridden-day-types (overridden-day-type*)>

<!ELEMENT overridden-day-type (interval*)>
<!ATTLIST overridden-day-type id         CDATA #REQUIRED>

<!ELEMENT days (day*)>
<!ELEMENT day (interval*)>
<!ATTLIST day date                       CDATA #REQUIRED
              type                       CDATA #REQUIRED
              id                         CDATA #IMPLIED>
```

Appendix D

Formal Verification of Project Data

This appendix describes and compares an alternative data modelling approach that we could have used to construct a PM–to–PMM mapping. Our approach, based on the meta–relationship, can be used to construct XML schema files that test the conformance of project data to a PMM. We describe an XSLT transform that translates PMM XML files, written using TPMM, to XML schema files that can perform this verification function.

D.1 An Alternative Approach to modelling PMMs

It is possible to take an approach to modelling the "Project/PMM Universe" that is strictly based on generalization [80], or the meta–relationship. This approach is used for the Ripple metadata tool [92], for instance. By following a path of meta–relationships, we can derive a data model for recording project data from a PMM description. In terms of the object–oriented model: object instances at one level become class definitions for the level below or, in terms of relational tables table rows in the PMM description become table headings for recording project data. The result of this abstraction is a project data model specific to the PMM, which records project data in terms defined by the PMM used to produce it. This approach was applied via XML, as follows:

1. PMMs are described using XML Schema: PMM steps (sequence, series and repetition), control structures and resource requirements are described.

2. Project tasks are recorded as XML data, constrained by the PMM XML schema to conform to the PMM, with start/stop times and resources assigned.

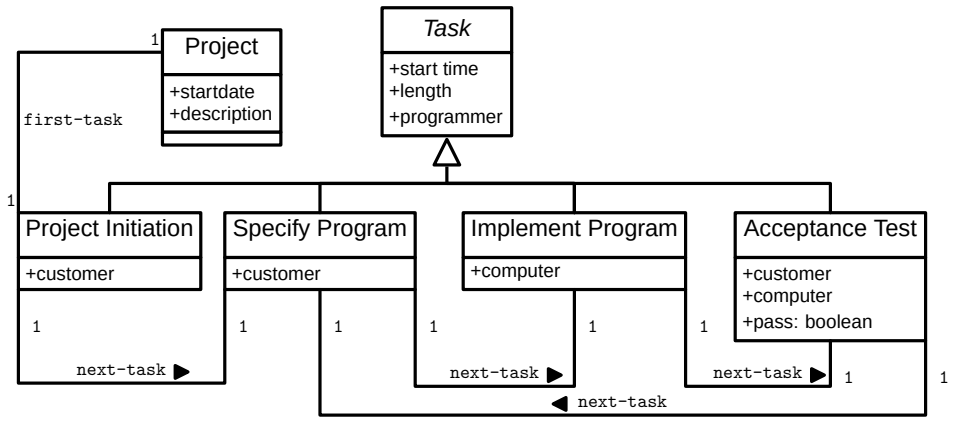


Figure D.1: *UML diagram of the Build–and–Fix method represented directly as classes.*

For example, we produced the UML model of figure D.1 by abstracting a project instance from the Build–and–Fix PMM:

1. The customer and the programmer initiate the project, and then

2. Iterate the following sequence of steps until the customer accepts the program or halts the project:

   (a) The customer specifies the program to the programmer

   (b) The programmer writes the program with a computer

   (c) The programmer and the customer test the program with a computer

Instead of specifying steps, figure D.1 specifies tasks. We represent the semantics of the Build and Fix PMM with the "next–task" relation that constrains the sequence of of the PMM–specified tasks, "Specify, Implement, Acceptance" This model could be translated to an XML schema, or to relational tables, using the strategies of chapter 7. Another expression of this data model is as printed forms, and suitable forms, specifically tailored for recording the Build–and–Fix PMM, are shown in table D.1. The form has a number of "N.A." fields that are not to be used, but are an artifact of storing different step types together. This is done to constrain the order of the tasks.

Build and Fix Project

Step Name	start	length	Customer Name	Programmer Name
Project Initiation				

Step Name	start	length	Cust. Name	Prog. Name	Com-puter	Pass (Y/N)
Specify Program					**N.A.**	**N.A.**
Write Program			**N.A.**			**N.A.**
Accep-tance Test						
Specify Program					**N.A.**	**N.A.**
Write Program			**N.A.**			**N.A.**
Accep-tance Test						
(etc. . .)						

Table D.1: *Example project record sheets for the Build–and–Fix PMM created using the approach of section D.1. The forms intrinsically constrain which tasks can be recorded to conform to the PMM.*

Concept	Data–oriented	Process–oriented
PMM description language	one	one
Project description language	Many specific schemata	One general schema
Relationship between the two languages	Meta–relationship	Process instantiation relationship
Cardinality of above relationship	one PMM language to many project languages	one PMM language to one project language
Relationship between project and PMM data	PMM is intrinsic in the project description, so that project data can be recorded without reference to the PMM	PMM description is required to create and interpret project data.
Instantiation of PMM	PMM is instantiated as a schema	PMM is instantiated as a process
Project description integrity check	schema–oriented; impossible to record faulty data.	Process–oriented; a process must check project data integrity against PMM description.

Table D.2: *Comparing the two data modelling approaches*

The meta–relationship approach produces data models which are specific to particular PMMs, so that PMM semantics and associated software tools are specific to a particular PMM, and the the project instance data intrinsically has a particular PMM associated with it. Such an approach is used by the Maven [51][41] project tool set, and so a software project must be "Maven enabled" before the tools can be used on it. Maven automates such practices as unit testing, CVS checkin and checkout of code, etc—but does not allow alternative practices for software engineering, so it would be impossible to use Maven for the Build–and–Fix PMM, for example. Table D.2 summarizes the differences between the two approaches.

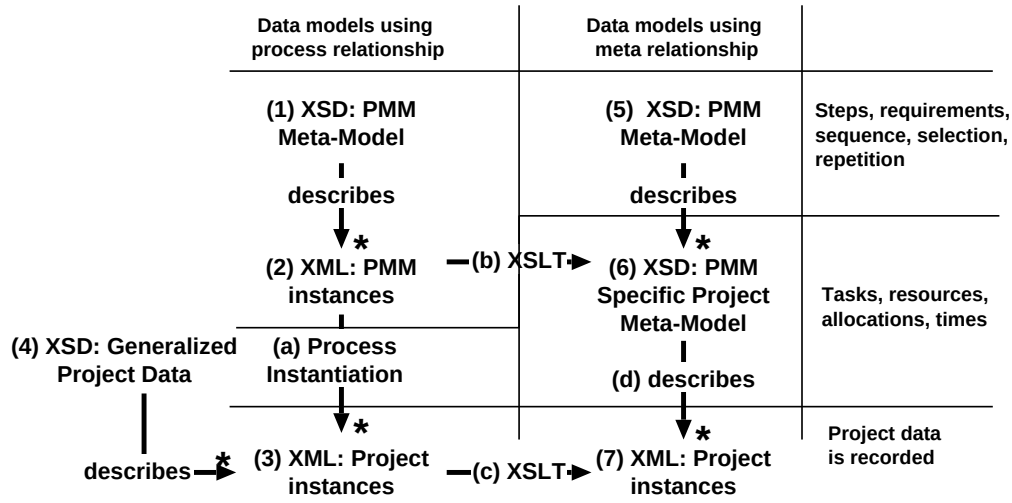


Figure D.2: *Relating the two data modelling approaches, how they map PMMs to project instances, and how to map one to the other. The models are described either with XSD (XML schemata), or XML defined by those schemata. The* $\longrightarrow^*$ *symbols depict a one–to–many mapping.*

D.2 XML Schematic Verification of PMM project data

Using the process–oriented data modelling approach, we have created a general XML schema for describing PMM data; figure D.2 (1). Data described by this model; figure D.2 (2), can be instantiated using a process; figure D.2 (a), to produce project data in terms defined by a generalized schema; figure D.2 (4). The process maps PMM semantics; steps, requirements, etc, to attributes in the project data; figure D.2 (3).

Compare this to the meta–relationship data modelling approach of section D.1: we create PMM descriptions using XSD as the PMM language, shown in figure D.2 (6). This schema can itself be constrained by a schema; figure D.2 (5), that restricts the XSD language: PMM sequences, for instance, constrain tasks to a specified sequence.

The process–oriented approach abstracts both the PMM and the project data models, so that in object–oriented terms, the difference between this approach and the process–oriented approach to PMM descriptions is this: The process–oriented approach relates PMM steps to project tasks using an attribute on each task that records the PMM step which created it, because

the project data model is a general one. By comparison, the metadata approach relates PMM steps to project tasks by naming each (PMM–specified) task class, because every PMM creates a different model for project data.

We used the alternative approach described above to verify project data produced using the PM/PMM framework. We wrote an XSLT transform; "pmmtoxsd.xsl", to implement the transformation of figure D.2 (b), that changes a PMM description from the process–oriented model used by Project Mentor to the schema–oriented model. The resultant XML schema; figure D.2 (6), constrained project data; figure D.2 (7), to conform to a particular PMM, that was now embedded in the schema. Also embedded in this schema are project concepts of tasks, resources, allocations and times.

The project data produced by Project Mentor, that realises figure D.2 (a), conforms to a generalized data model; figure D.2 (4). In our implementation, the data model used is the Planner XML schema, augmented by custom properties to contain the PMM–specific attributes, such as the PMM step associated with each task. An XSLT transform to implement figure D.2 (c); "mrptopmm.xsl", was written to transform the Planner data into XML conforming to the XML schemata of figure D.2 (6). The XML project data produced by Project Mentor was then schematically verified against this generated XML schemata for a number of PMMs, figure D.2 (d). The sequence of steps used was:

1. Create a PMM description *P.XML* using the TPMM schema.

2. Transform *P.XML* to schema *P.XSD* using pmmtoxsd.xsl.

3. Use Project Mentor to create some project data *D.XML* using the PMM description *P.XML*.

4. Transform *D.XML* to *D.XML'* using mrptopmm.xsl.

5. Edit the XML files to set XML namespaces correctly.

6. Use *P.XSD* to verify *D.XML'*.

Using this approach, it was possible to verify that project data produced with Project Mentor conformed to an XML schema that was generated from the PMM that Project Mentor used. A number of logic errors in Project Mentor were brought to light by this exercise: project tasks were wrongly ordered in some circumstances, for instance. The result was that the *D.XML'* file produced in these cases did not verify against the *P.XSD* schema.

Appendix E

CDROM insert

This appendix contains a CDROM, the contents of which are:

- Listings of the XML schemata used by Project Mentor, the implementation of our framework.

- An HTML page with the Project Mentor applet ready to run with the above PMMs and visualizations.
- The source code for Project Mentor.
- A small library of Project Management Methods written in terms of our framework, using our TPMM schema.
- Nassi–Shneiderman visualizations of the PMMs.
- Files used for our survey and our case study.

References

[1] Abran, A., Moore, J., Bourque, P., and Dupuis, R., Eds. *Software Engineering Body of Knowledge*, trial version 1.00 ed. IEEE Computer Society, Los Alamitos, California, May 2001.

[2] Agerskov, C. Open Project Management Exchange Format. In *http://www.opmef.org/* (2005).

[3] Ahmed, K., Ayers, D., Birbeck, M., Cousings, J., Dodds, D., Lubell, J., Miloslav, N., Rivers-Moore, D., Watt, A., Worden, R., and Wrightson, A. *Professional XML Meta Data*. Programmer to Programmer. Wrox Press, USA, `http://www.wrox.com`, 2001.

[4] Batik. A Java technology based toolkit for Scalable Vector Graphics. In *http://xmlgraphics.apache.org/batik/* (2006).

[5] Beck, K. *Extreme Programming Explained: Embrace Change*, 6th ed. XP series. Addison-Wesley, 2000.

[6] Boehm, C., and Jacopini, G. Flow Diagrams, Turing Machines and Languages with only Two Formation Rules. *Communications of the ACM* (May 1966), 366–371.

[7] BPEL4WS. Business Process Execution Language for Web Services. In *ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf* (May 2003), BEA Systems, IBM, Microsoft, SAP AG and Siebel Systems.

[8] BPML. Business Process Markup Language. In *http://www.bpmi.org/* (Aug. 2000), BPMI Initiative.

[9] Bradley, N. *The XML companion*, third ed. Addison–Wesley, London, 2002.

[10] Brooks, F. P. *The Mythical Man Month*, 1995: anniversary ed. Addison–Wesley, Reading, Mass.

[11] Burbridge, R. N. G. Introduction. In *Perspectives on Project Management*, R. N. G. Burbridge, Ed., IEE Management of Technology Series No 7. IEE, London, 1988, pp. xv–xxv.

[12] Bussler, C. Workflow Instance Scheduling with Project Management Tools. In *9th International Workshop on Database and Expert Systems Applications (DEXA'98)* (`http://doi.ieeecomputersociety.org/10.1109/DEXA.1998.707492`, 1998), IEEE Computer Society, p. 753.

[13] Codd, E. F. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM 13*, 6 (1970), 377–387.

[14] Collins, T., and Bicknell, D. *Crash, Ten Easy Ways to Avoid a Computer Disaster*. Simon and Schuster, 1997.

[15] CORBA. Common Object Request Broker Architecture. In *`http://www.corba.org/`* (2006), The Object Management Group.

[16] Crewdson, T. *INCIS Project Business Case*. NZ Police, 12 May 1993. Schedule 5 in [79].

[17] The Critical Path Problem. In *Dictionary of Algorithms and Data Structures*, P. E. Black, Ed. `http://www.nist.gov/dads/HTML/criticalPath.html`, 2004.

[18] Dale, A., Churcher, N., and Irwin, W. A Framework for Linking Projects and Project Management Methods. In *PROFES2005* (`http://dx.doi.org/10.1007/11497455_9`, June 14-15 2005), F. Bomarius and S. Komi-Sirvio, Eds., vol. 3547/2005, Springer, pp. 84–97.

[19] Dale, A., and Goldfinch, S. Pessimism as an IS Management Tool in the Public Sector: Lessons from the INCIS Fiasco in the New Zealand Police Force. In *http://www.cosc.canterbury.ac.nz/research/reports/TechReps/2002/tr_0202.pdf* (2002).

[20] Dale, T. Project Mentor: A Tool to Control Projects With Project Management Methods. In *http://www.cosc.canterbury.ac.nz/tony.dale/msc/pmentor* (2005).

[21] Dale, T. TPMM and PM/PMM framework. In *http://www.cosc.canterbury.ac.nz/tony.dale/msc/index.html* (2005).

[22] Dale, T. Alterations to the Planner XSL transforms to support Nassi–Shneiderman visualization. In *http://www.cosc.canterbury.ac.nz/tony.dale/msc/planner/* (2006).

[23] Dale, T. pmmtosvg; an XSL transform to convert PMMs to Scalable Vector Graphics. In *http://www.cosc.canterbury.ac.nz/tony.dale/msc/pmentor/pmmtosvg.xsl* (2006).

[24] DeGrace, P. *Wicked Problems, Righteous Solutions. A catalogue of modern software engineering paradigms.* Prentice–Hall, Upper Saddle River, NJ, 1991.

[25] Dijkstra, E. W. Go To Statement Considered Harmful. *Commmunications of the ACM (Letters to the Editor) 11*, 2 (1968), 147–48.

[26] Department of Defence Instruction 5000.2. In *http://akss.dau.mil/dapc/index.html* (2002), Dept of Defence, USA.

[27] DOM. Document Object Model Level 3 Core Specification. In *http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/* (Apr. 2004), The W3C Consortium.

[28] DTD. Document Type Definition. In *http://en.wikipedia.org/wiki/Document_Type_Definition*.

[29] Duckett, J., Griffin, O., Mohr, S., Norton, F., Stokes-Rees, I., Williams, K., Cagle, K., Ozu, N., and Tennison, J. *Professional XML Schemas*. Programmer to Programmer. Wrox Press, USA, `http://www.wrox.com`, 2001.

[30] ECMAScript Language Specification. In *`http://www.ecma-international.org/publications/standards/Ecma-262.htm`* (Dec. 1999), ECMA International.

[31] Fayol, H. *General and industrial management*, revised by Irwin Gray ed. IEEE, New York, 1984. A translation of "Administration Industrielle et Generale", published under the sponsorship of the IEEE Engineering Management Society.

[32] Fenton, N., and Pfleeger, S. *Software Metrics, A Rigorous & Practical Approach*, 2nd ed. PWS Publishing Company, Boston, MA, 1997.

[33] Flowers, S. *Software Failure, Management Failure: Amazing Stories and Cautionary Tales*. Wiley, Chichester; New York, 1996.

[34] Fowler, M. *Refactoring: Improving the Design of Existing Code.* The Addison-Wesley object technology series. Addison Wesley, USA, 1999.

[35] Fowler, M., and Scott, K. *UML distilled: a brief guide to the standard object modeling language*, 2nd ed. Addison Wesley, USA, 2000.

[36] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object–Oriented Software*. Addison–Wesley, Reading, Massachusetts, 1995.

[37] Gantt, H. L. *Organizing for Work*. George Allen & Unwin Ltd, London, UK, 1919.

[38] Gauld, R., and Goldfinch, S. with Dale, T. *Dangerous Enthusiasms: E–Government, Computer Failure and Information System Development*. Otago University Press, Dunedin, New Zealand, 2006.

[39] Gray, C. F., and Larson, E. W., Eds. *Project Management, The Managerial Process*. McGraw–Hill, New York, 2000.

[40] Hawkins, G. *$60 million to address years of property neglect*. Ministerial press release `http://www.executive.govt.nz/speech.cfm?speechralph=35207&SR=0`, July 2nd 2001.

[41] Hightower, R., Lesiecki, N., and Zawadzki, M. *Professional Java tools for Extreme Programming: Ant, Xdoclet, JUnit, Cactus and Maven*. Wiley Technology, 2004.

[42] Hult, R., Hallendal, M., and del Castillo, A. *Planner, a Project Management Application for the Gnome Desktop*. `http://www.imendio.org/projects/planner/`, 2004.

[43] Ifergan, S. S., Maarek, Y. S., and Ur, S. Xeena, a visual XML editor. In *http://www.alphaworks.ibm.com/tech/xeena* (2004).

[44] ISERC. Agile Approaches. In *http://www.iserc.ie/AgileApproaches.html* (2004), Irish Software Engineering Research Consortium.

[45] JAXB. The Java Architecture for XML Binding. In *http://java.sun.com/webservices/jaxb/about.html* (Oct. 2004), Sun Microsystems.

[46] Jeffries, R., Anderson, A., and Hendrickson, C. *Extreme Programming Installed*. XP series. Addison-Wesley, 2001.

[47] Kelly, J. E., and Walker, M. R. *The Critical Path Method*. Remington Rand and DuPont Corporation, 1957.

[48] Lightweight Directory Access Protocol; LDAP. In *http://www.openldap.org/*, The OpenLDAP project.

[49] Lewis, J. *Fundamentals of Project Management*. AMACOM, New York, 1995.

[50] Maurer, F., Dellen, B., Bendeck, F., Goldmann, S., Holz, H., Kotting, B., and Schaaf, M. Merging Project Planning and Web-Enabled Dynamic Workflow Technologies. *IEEE Internet Computing* (May/June 2000), 65–74.

[51] Maven Project Management Tool. In *http://maven.apache.org/* (May 2004), The Apache Project.

[52] McConnell, S. *Rapid Development: Taming Wild Software Schedules*. Microsoft Press, Redmond, Wa., 1996.

[53] Microsoft. Creating Microsoft Project 2002 Reports Using XML and XSL. In *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/pdr/PDR_ProjXML_3352.asp* (2003), MSDN Library.

[54] Microsoft. Extracting Timephased Data from the Microsoft Project Database. In *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/pdr/PDR_Timephased_Data_3337.asp* (2003), MSDN Library.

[55] Microsoft. Microsoft Project 2003. In *http://www.microsoft.com/* (2003), Microsoft Corporation.

[56] Microsoft. Microsoft Project Server Data Reporting. In *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/pdr/PDR_Data_Reporting_3355.asp* (2003), MSDN Library.

[57] MIL-HDBK-881: Handbook Work Breakdown Structure. In *http://www.acq.osd.mil/pm/currentpolicy/wbs/mil_hdbk_881/mil_hdbk_881.htm* (1998), Dept of Defence, USA.

[58] MSPDI. Microsoft Office 2003 XML schemata. In *http://www.microsoft.com/downloads/details.aspx?FamilyID=*

fe118952-3547-420a-a412-00a2662442d9&DisplayLang=en (14 January 2005), Microsoft Corporation.

[59] Nassi, I., and Shneiderman, B. Flowchart Techniques for Structured Programming. *Sigplan 8*, 8 (1973), 12–26.

[60] *AS/NZS 4360: Risk Management*. Standards New Zealand, 1999.

[61] O'Connor, A. J., and Ganong, G. H. D. Some Thoughts on High Budget Projects. In *Perspectives on project management*, R. N. G. Burbridge, Ed., IEE Management of Technology Series No 7. IEE, London, 1988, pp. 22–33.

[62] Oechsle, R., and Schmitt, T. JAVAVIS: Automatic Program Visualization with Object and Sequence Diagrams Using the Java Debug Interface (JDI). In *Software Visualization: international seminar, Dagstuhl castle, Germany*, S. Diehl, Ed., fourth ed., Lecture Notes in Computer Science. Springer, Heidelberg, Germany, 2002, pp. 176–190.

[63] O'Neill, T. W. IBM Rational Project Tracker Informational Guide. In *http://www-128.ibm.com/developerworks/rational/library/05/0809_ONeill/* (2005).

[64] Open Software. *The Vim Text Editor*. `http://www.vim.org/`.

[65] Open Software. Docbook Authoring Tools. In *http://wiki.docbook.org/topic/DocBookAuthoringTools* (2006).

[66] *PERT (Program Evaluation and Review Technique)*. Navy Special Projects Office and Booz, Allen, and Hamilton consultants, Jan. 1958.

[67] PMI. *Project Management Body of Knowledge*. Project Management Institute, Inc, Four Campus Blvd, Newtown Square, Pennsylvania, USA, 2000.

[68] PMXML. Project Management XML Schema. In *http://www.projectoffice.com/xml/* (Jan. 2000), Pacific Edge Software.

[69] Price, B. A., Baecker, R. M., and Small, I. S. A Principled Taxonomy of Software Visualization. *Journal of Visual Languages and Computing 4*, 3 (1993), 211–66.

[70] RAE and BCS. *The Challenges of Complex IT Projects.* The Royal Academy of Engineering, `www.raeng.org.uk`, 29 Great Peter Street, Westminster, London, SW1P 3LW, Apr. 2004. The report of a working group from the Royal Academy of Engineering and the British Computer Society.

[71] Rational Unified Process: Best Practices for Software Development Teams. In *http://www-128.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf* (1998), TP026B, Rev 11/01.

[72] Roling, G., and Freisleben, B. Experiences In Using Animations in Introductory Computer Science Lectures. In *SIGCSE 2000* (2000).

[73] Royce, W. W. Managing the Development of Large Software Systems: Concepts and Techniques. In *IEEE WESTCON* (Los Angeles, 1970), pp. 1–9.

[74] Rozman, T., Horvat, R., and *Polančič*, G. Towards True Process Descriptions Interoperability. *Journal of Computing and Information Technology CIT 12*, 2 (2004), 151–157.

[75] Russell, N., van der Aalst, W., ter Hofstede, A., and Edmond, D. Workflow Resource Patterns: Identification, Representation and Tool Support. In *CAISE 2005* (Porto, Portugal, 2005).

[76] Schach, S. R., Ed. *Object–Oriented and Classical Software Engineering.* McGraw–Hill, New York, 2002.

[77] Schachter, V. How Does Concurrency Extend the Paradigm of Computation? *The Monist 82*, 1 (Jan. 1999), 37–57.

[78] Shapiro, R. A Comparison of XPDL and BPML and BPEL4WS. In *http://www.ebpml.org/A_Comparison_of_XPDL_and_BPML_BPEL.doc* (Aug. 2002), Cape Visions Ltd.

[79] Small, D. F. Ministerial Inquiry into INCIS. In *http://www.justice.govt.nz/pubs/reports/2000/incis_rpt/index.html* (Wellington, Oct. 2000), NZ Justice Department.

[80] Smith, J., and Smith, D. Database Abstractions: Aggregation and Generalization. *ACM Transactions on Database Systems 2*, 2 (June 1977), 105–133.

[81] The Chaos Chronicles. In *http://www.standishgroup.com* (1994), The Standish Group.

[82] Extreme Chaos. In *http://www.standishgroup.com* (2001), The Standish Group.

[83] SVG. Scalable Vector Graphics Full 1.2 Specification. In *http://www.w3.org/TR/SVG12/* (Apr. 2005), The W3C Consortium.

[84] Thomas, A., and Barashev, D. GanttProject, a Project Management Tool. In *http://ganttproject.sourceforge.net/* (2006), Sourceforge.

[85] US Air Force. *Performance Measure for Selected Acquisitions*. US Department of Defence Instruction 7000.2, 1967.

[86] van der Aalst, W. Patterns and XPDL: A Critical Evaluation of the XML Process Definition Language. `http://www.citi.qut.edu.au/pubs/ce-xpdl.pdf`.

[87] Volz, R. PMXML—a XML standard for project management. In *http://www.vrtprj.com/content/istandards/pmxml_en.html* (June 1st 2002).

[88] Waitai, R. *Inquiry into CARD and INCIS*. Justice and Law Reform Committee, Wellington, Oct. 1999.

[89] Westfechtel, B. *Models and Tools for Managing Development Processes*, vol. 1646 of *Lecture Notes in Computer Science*. Springer, Berlin, London, 1999.

[90] Wiggins, M. An Overview of Program Visualization Tools and Systems. In *Proceedings of SIGCSE* (1998), ACM Press, pp. 194–200.

[91] Williams, M. M. Extending the Frontier of the Extreme Programming Software Engineering Process. Master's thesis, University of Canterbury, `http://www.cosc.canterbury.ac.nz/research/reports/MastTheses/2004/mast_0402.pdf`, 2004.

[92] Wilson, R. P. RIPPLE : A metadata repository. Master's thesis, University of Canterbury, Computer Science Department, 1992.

[93] XML. Extensible Markup Language: XML. In *http://www.w3.org/xml* (Feb. 1998).

[94] XPDL. XML Process Definition Language. In *http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf* (Oct. 2002), Workflow Management Coalition.

[95] XSD. XML Schema Definition. In *http://www.w3.org/2001/XMLSchema/* (Feb. 2001), The W3C Consortium.

[96] XSL. Extensible Stylesheet Language. In *http://www.w3.org/Style/XSL/* (1999), The W3C Consortium.

[97] zur Muehlen, M. Resource Modeling in Workflow Applications. In *http://www.workflow-research.de/Publications/PDF/MIZU-WF99.PDF* (1999), Technical Report, University of Muenster, Germany.