

A Framework for Linking Projects and Project Management Methods

Tony Dale, Neville Churcher, and Warwick Irwin

Software Engineering and Visualisation Group,
Computer Science and Software Engineering Department,
University of Canterbury

{Tony.Dale, Neville.Churcher, Warwick.Irwin}@canterbury.ac.nz
<http://www.cosc.canterbury.ac.nz/research/RG/svg/index.html>

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.

We present data models for PM and PMM, in a framework that can articulate the PM-to-PMM relationship, illustrating with simple examples. A java/XML implementation of this framework can create and then revise a “PMM-aware” project, conforming to a specified PMM. In terms of the framework, we describe a simple project data visualization and associated method that can be used to synthesize a PMM for a project instance that was initially created without reference to any PMM.

Keywords: Project management, software engineering, project management methods, project processes, data modelling, XML, visualization.

1 Introduction

Conventional PM attempts to manage tasks and resources as closely as possible to a predefined, static plan.

However, a project plan doesn't indicate how its particular tasks or resources were created, except perhaps for a textual description. This is because the PM domain 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 or changing a project plan. Applying PMMs and process concepts to conventional PM creates a fundamental problem: PMMs dynamically change projects, but conventional PM is static, and has difficulty tolerating change.

It is possible to create data models for both PM and PMMs, and to link these models together in a unified framework. This framework has the power to express complicated PMM ideas but is simple and logical to apply to PM data: project tasks and resources have PMM concepts added to them as simple

attributes, for example. This strategy makes it possible to move backwards and forwards between the PM and PMM domains, so that a project can be created and changed according to a PMM, yet viewed with existing PM software tools.

The motivation for creating this framework is to provide a bridge between the process-dominated world of PMMs and the plan-dominated world of PM. We hope to encourage the more widespread use of process concepts and PMMs in conventional PM, and to augment PMMs with the project history and context available using conventional PM tools.

The rest of the paper is set out as follows: Section 2 introduces concepts of conventional project management (PM) as it is widely practised, and introduces a data model for PM software tools. We show in Sect. 3 how the specialist knowledge that all projects make use of can be formalized as a Project Management Method (PMM), and create a data model for describing PMMs in Sect. 4. In Sect. 5 we create a framework for linking the PM and PMM data models, and using this framework Sect. 6 illustrates how a PMM is applied to produce a project instance. Section 7 uses the framework to create a simple visualization and a heuristic method to derive a PMM from raw project data, and illustrates a scenario for applying process improvement to the derived PMM. Section 8 concludes our discussion.

2 Background

Large-scale project management has been practised for centuries, to the extent that Burbridge [1] says

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.

A crucial change has been, from the 19th century onwards, the increasing importance of time and cost in project management. Projects must be completed on time and within budget, according to comprehensive project plans, for reasons such as increasing company profitability or reducing expenditure of public funds. Unfortunately, the failure rate of projects in certain fields such as Software Engineering is very high, with the Standish Group reporting in 2001 that only 28% of IT projects completed on time and within budget [2], although this is an increase from the 16% of 1994 [3]. There is thus a motivation to provide more powerful tools for project management.

Conventional project management is encapsulated in the following five steps, as exemplified by Lewis [4]:

1. define the problem
2. plan the project
3. execute the plan
4. monitor and control progress against the plan
5. close the project.

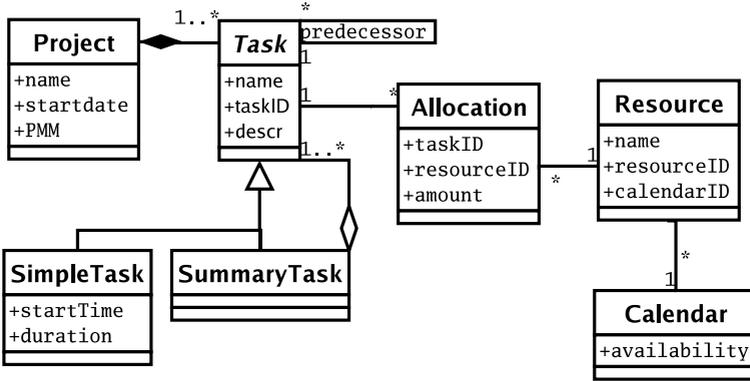


Fig. 1. A conceptual data model of conventional PM data: a project is made up of Tasks, Resources and Allocations of tasks to resources

These steps provide a simple framework for creating and executing projects using a static, unchanging plan. Such a plan can be produced from a template that has been predefined for a particular goal. For example: Microsoft Project [5] provides a template for software development. Carefully filling in this template will result in a project plan for developing software that looks quite plausible. However, there is no indication whether the project is a sure-fire success or doomed from the start, and nothing to tell us what to do if the actual progress begins to deviate from the plan. This is not a problem specific to Microsoft Project: the problem is with the domain of conventional PM in which this software tool works. Conventional PM lacks the capability to say why the tasks and resources in the Microsoft Project templates are there, except with textual descriptions. We cannot say what created the templates, or compare them. This lack of descriptive ability is one of several reasons that change in a project plan is regarded as a bad thing by conventional PM, and uncontrolled change in a project is identified as a major cause of many software project disasters [6][7][8].

Once a project is started, conventional Project Management is concerned with monitoring the progress of a project using various well-known methods to measure and report on the project, such as the Critical Path method, Gantt charts, Network charts and Earned Value Analysis. These are all methods aimed at keeping the progress of a project as close as possible to a static, pre-defined plan.

Project Management software such as Microsoft Project and Planner [9] support the management and measurement of conventional PM, and provide templates for project plans. PM software tends to be variations on a theme, so that Gray [10] writes of PM software tools: "Differences among [PM] software in the last decade have centered on improving 'friendliness' and output that is clear and easy to understand."

Each tool has its own data format, but there are common features so that, for example: Microsoft Project and Planner can exchange data using XML. Analysis

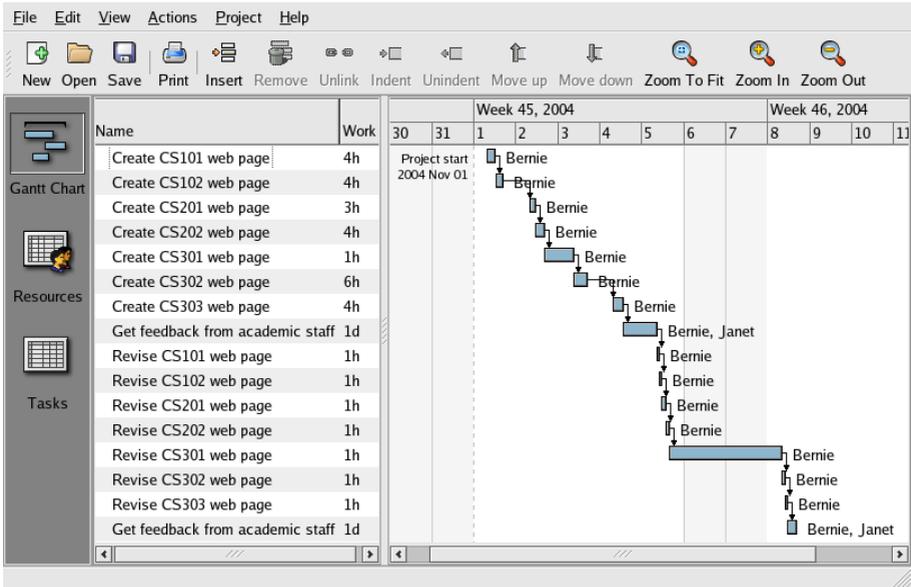


Fig. 2. A conventional PM view of a project: the free software tool Planner is illustrating project data, as modelled in Fig. 1, with a Gantt [13] chart

of the facilities and data models used for a number of project tools, including Microsoft Project, the freeware tools Planner and PyGantt [11] and the public XML schema PMXML [12] shows that their data models reflect and record tasks, resources and allocations of resources to tasks. This data model is summarized in a conceptual way in Fig. 1. Figure 2 instantiates this data model, and shows a typical view of a project presented by conventional PM tools.

3 Project Management Methods (PMMs)

Any real project occurs in a specialist domain of some kind, whether it is bridge construction, software engineering or a company restructuring. A real project therefore requires specialist knowledge to inform the planning and implementation process. For example: estimating the time a particular task will take requires specialist knowledge of the work involved in the task.

Conventional PM has little to say about how to use this specialist knowledge in a project: conventional PM methods for representing and applying specialist project knowledge stop at the the use of predefined project templates, and the advice Lewis offers for task estimating is that you will get better at it as you do more of it. Often, specialist knowledge is applied to a project in an informal way, using whatever skills a project manager may bring to bear.

Specialist project knowledge can be formally described and applied using a Project Management Method (PMM). A PMM is a notation for turning special-

ist knowledge into a “project recipe”: a list of steps and requirements to follow in order to achieve a project goal in a particular area, such as software engineering.

A simple way to describe PMMs is to write them down in English as a series of steps. As an example, here is a high-level description of the “Build and Fix” method for writing programs. Schach [14] describes this simple method as suitable for small programs.

1. The customer and the programmer initiate the project, and then
2. Iterate the following sequence of steps until the customer accepts the program:
 - (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

A process description or PMM like the one listed above provides a means of decomposition of a large task (the project goal) into sub-tasks, along with an indication of resource requirements. For example: if we apply the above PMM to the goal of creating a “Hello World” program, we initially create this project instance:

Step 1: Initiate the “Hello World” project. *Requires a Customer*

Step 2(a): Specify the “Hello World” program. *Requires a Customer and a Programmer*

Step 2(b): Write the “Hello World” program. *Requires a Programmer and a computer*

Step 2(c): Test the “Hello World” program. *Requires a Customer, a Programmer and a computer*

At this point, we have reached the task of testing the program, Step 2(c), and we use the result of this test to decide whether to repeat step 2: suppose that the customer is dissatisfied with the program. We then have to start step 2 again, so that the project instance might become:

Step 1: Initiate the “Hello World” project. *Requires a Customer*

Step 2(a): Specify the “Hello World” program. *Requires a Customer and a Programmer*

Step 2(b): Write the “Hello World” program. *Requires a Programmer and a computer*

Step 2(c): Test the “Hello World” program. *Requires a Customer, a Programmer and a computer*

Step 2(a): Specify the “Hello World” program. *Requires a Customer and a Programmer*

Step 2(b): Write the “Hello World” program. *Requires a Programmer and a computer*

Step 2(c): Test the “Hello World” program. *Requires a Customer, a Programmer and a computer*

If, after the second iteration of the PMM, our customer is satisfied with the program, the loop terminates, the PMM finishes and we are left with the above project instance.

We can see that a PMM specifies the production of a project instance or, conversely, a project instance may represent the “footprints” produced by the operation of a PMM when it is applied to a particular goal. When a PMM is applied, PMM steps are translated to project tasks and resource requirements are translated into project resources and allocations of resources to tasks.

If we were to apply conventional PM to the production of the “Hello World” program then we would produce a project plan which lacks any PMM information:

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 this case, we are in a quandary if 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 reimplementaion might be required, for instance. Changes like this are a problem characteristic of conventional PM: extra tasks, resources and allocations are almost always inserted into a real project, but the only coping strategy of conventional PM is to try to minimize change.

Because conventional PM lacks the formal concepts to deal with project change, software tools for conventional PM don’t support change very well: reports can be produced to indicate the degree of variance from the original plan, but the plan is not supposed to change. The result is that changes to a conventional project often happen in an uncontrolled way. The PMM-driven project at least specifies what changes should be made to the project instance, and why. The disadvantage of the PMM-driven project is that the lack of a static plan precludes a fixed-cost quote for a project, for instance. Iterative PMMs such as XP [15] 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 deliverables are not fixed in advance.

4 Modelling PMMs

We examined a number of Software Engineering PMMs including the Waterfall process [16] and Extreme Programming, and workflow languages such as BPML [17]. Although some PMMs were essentially sequential and others essentially iterative, we found that the three constructs identified by Boehm and

Jacopini [18]: sequence, selection and repetition, could describe the flow of control in all PMMs. We also used a container step, the `UnSequencedStep`, for parallel activities, to synchronizes parallel flows of control. We can see in our example PMM of Sect. 3 that these constructs are sufficient to describe the flow of control of the Build-and-Fix PMM: The main control structure of this PMM is repetition (step 2) that generates a series of implement/test sequences (steps 2(a)–2(c)) in the resulting project instance.

The control structures of a PMM constitute the fundamental difference with conventional PM: the result is that PMM steps may be repeated, or not used at all, whereas every task in a conventional PM project plan is expected to be actioned exactly once. The concepts of control flow, that create and change project instances, are not represented in conventional project management techniques, as supported by common PM software tools.

To describe PMMs and allow their programmatic representation, we have used the concepts above to design the conceptual data model in the upper left quadrant of Fig. 3 (a). By describing PMMs in this way, a number of advantages accrue: we can describe PMMs, and compare them. We can apply PMMs in a repeatable way, and we can describe the relationship between PMMs and project instances. This model relates in a straightforward way to existing process modelling architectures. For example, the development process architecture of SUKITS [19] consists of a process instance level that is described by a process definition level, similar to Fig. 3 (a). We can instantiate the PMM data model in many different ways; using XML [20], Java classes, etc. For example, we can create an XML representation of the “Build and Fix” method of Sect. 3, shown abbreviated in the lower left quadrant of Fig. 3 (a). More complicated PMMs such as Extreme Programming can also be represented with this PMM data model.

5 The PM/PMM Framework

We have formulated conceptual data models for PM (Fig. 1) and PMM (Fig. 3 (a)). The example of Fig. 2 illustrates how the PM data model is instantiated: `Tasks`, `Resources` and `Allocations` are recorded to create a `Project`. The lower left quadrant of Fig. 3 (a) instantiates the PMM data model to record the “Build and Fix” PMM description.

We now tie these two data models together to form a unified framework, as shown in Fig. 3: The PMM data model (a) is related to the conventional Project data model (b) by the mappings `applied--to--goal`, `describes` and `role`. To record these mappings to relate as “PMM-aware” project data, we have added the following attributes to the PM model of Fig. 1, to create an augmented PM model, shown in Fig. 3 (b): The `Task` class gains a `stepID` attribute, the `Project` class gains a `PMM` attribute and the `Resource` class gains a `resourceTypeID` attribute.

Because our framework relates PMMs to PM, we can make use of the complementary strengths of both areas. Conventional PM can provide a useful “big

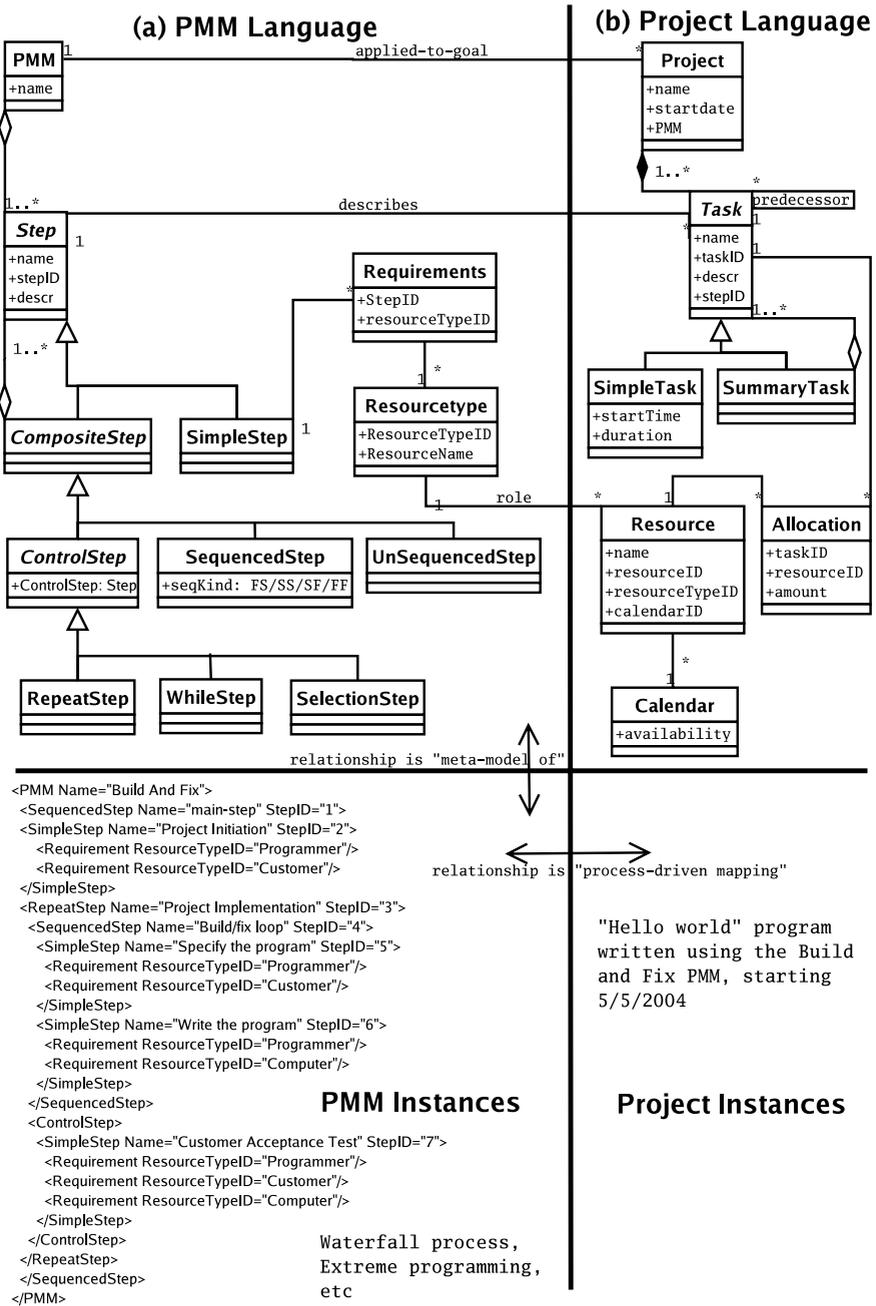


Fig. 3. UML diagram showing the PMM/PM framework, with instances. A feature of the PMM and PM data models is the Composite design pattern [21], that provides a tree-structured decomposition of objects, useful for XML representation

picture” view of a project, because PM tools provide the context of a project history, and a (projected) project future. PMMs do not intrinsically provide such a project context, but if the current state of a PMM can be viewed in the context of the project history, as a Gantt chart makes possible, for instance, then much more context is available than in a solely process-oriented view. Using our framework, this “big picture” view of a project is available with conventional PM tools, whilst still making use of PMM concepts. The connection from PMMs to conventional PM makes it possible to use PMMs for attempts to predict the future course of a project, and for scenario comparisons. Since predicting the future is what conventional PM attempts to do when planning a project, the use of a PMM to produce the project plan at least provides some systematic basis for the predictions. For example: the use of different PMMs could be compared, when applied to the same goal.

The framework also provides new methods for improving project processes: if we change a PMM project instance, say: by reversing the order of two tasks, then the project instance no longer fits the PMM that produced it. One way of reconciling the changes might be to change the order of the associated steps in the PMM. In this way we could use conventional PM tools to alter PMMs and tailor them for more specific applications.

6 Implementing the Framework

A Java/XML implementation [22] was written to test the framework of Fig. 3. The program manipulates two Java DOM [23] trees: The first DOM tree represents the PMM being instantiated. The XML Schema [24] that describes this DOM tree is a straightforward implementation of Fig. 3 (a). The second DOM tree records the tasks, resources and allocations produced by the PMM instance. The XML schema for this DOM tree is that of the freeware Planner [9] program. Similar to other PM software tools, the planner schema allows for custom properties to be added to projects, tasks and resources, and so these were used to record, for example: the PMM used for the project (the `PMM` attribute added to the `Project` class) and which PMM step was associated with each project task (the `stepID` attribute added to the `Task` class).

The mappings described in Sect. 5 were implemented in Java code as follows:

applied-to-goal. When a PMM is applied to a goal, these attributes are set in the `Project` instance:

- `start date` defaults to the current time, but may be changed with ordinary PM tools without impacting on the PMM process.
- `project goal` is set to the goal required to be accomplished.
- `PMM` is set to the PMM name used for the project.

describes. The steps of a PMM describe the tasks produced in the project instance. The kind of task produced depends on the kind of PMM step that describes it: a PMM `SimpleStep` produces a `SimpleTask` in the project instance: they contain a task name, start time and duration. Any kind of

CompositeStep, such as a **RepeatStep**, produces a **SummaryTask** in the project. Summary tasks, are tasks aggregated from other tasks or summary tasks, and are a common notation in projects. Summary tasks can be “rolled up” to hide the tasks they contain, so only the summary task is displayed by a software tool, for instance. Summary tasks derive their temporal information (eg: start time and duration) from the tasks they contain. In our framework, we make use of these features of summary tasks, so that PMM control step instantiation can be recorded in the project instance in such a way that no spurious temporal information is added to the project, and the hierarchical structure of the project data recorded reflects the structure of the PMM that produced the instance.

Attributes in the **SimpleTask** and **SummaryTask** instances are set as follows: **task name** is set from the PMM **Step** name.

start date is set to the start time of the project, plus the the sum of the task durations up to this point.

duration defaults to one day, but may be changed with ordinary PM tools without impacting on the PMM process.

stepID is set from the PMM **StepID**.

Our PMM model uses five kinds of **CompositeSteps** which are mapped to **SummaryTasks** in the following ways:

RepeatStep The PMM steps contained in a **RepeatStep** are instantiated as project tasks contained in a summary step, which is a predecessor of a **ControlStep**, also mapped to a project task. Continuation of the loop depends on whether the **ControlStep** task succeeded or failed: if the task succeeded, no further actions are taken. If the task failed then another group of project tasks and associated control step is created, and so on.

WhileStep This step works similarly to the **RepeatStep**, except that the **ControlStep** is instantiated first, and success or failure of that task determines whether to instantiate the steps contained in the **WhileStep**.

SelectionStep The **ControlStep** in a **SelectionStep** is instantiated as a project task that determines one and only one of the steps contained in the **SelectionStep** to be instantiated.

SequencedStep The PMM **SequencedStep** has the **seqKind** attribute to describe the kind of step sequence it contains: finish-to-start, finish-to-finish, start-to-finish, start-to-start (FS/FF/SF/SS). When a **SequencedStep** is instantiated, the steps it contains will be instantiated as a corresponding sequence of tasks with a **predecessor** relation, set to specify the kind of sequence (FS/FF/SF/SS).

UnSequencedStep The PMM steps contained in an **UnSequencedStep** are instantiated in the project instance within a summary step, with no predecessor information set.

role. The PMM describes what kind of resources are required with the **ResourceType** instance. When a new resource is created in the project instance, the **resourceName** attribute of this instance is mapped to the **name** attribute of the resource. When a PMM **Step** has a particular **requirement**,

the project instance is searched for resources having the same name (ie: the resource name is used as it's classification). If such a resource does not already exist then it is created in the project instance. The association of task to resource is then recorded in the `taskID` and `resourceID` attributes of an `Allocation` instance in the project.

At any time, the Java program can serialize the project DOM tree and write it out to a file, so that the “big picture” view of a PMM-driven project, described in Sect. 5, is available when the project instance is viewed with the Planner software tool. For viewing the project state in this way, it is convenient to serialize and then later re-instantiate an active PMM process, ie; to “pick up where you left off” with the PMM. To do this, a custom property, `Step-Nesting`, is used at serialization time to record in the project instance the state of the PMM process, along with the project data.

7 Synthesizing a PMM Using a Visualization and Heuristic

The java program of Sect. 6 allows us to instantiate a PMM to produce a project instance. However, examining a project instance that has been created without reference to a PMM in an attempt to synthesize a PMM which “fits” the project is difficult, because a potentially infinite number of PMMs might have created a particular project instance. A visualization may assist a human to deduce a PMM by examining project data, and using some heuristic method. Using the framework, we can create such a visualization, as follows:

1. Classify and group (using summary tasks) the project tasks and resources according to some scheme, so that we decorate the project instance with PMM “Kind of step” and “Kind of resource” data.
2. Decide whether particular groups of tasks are the result of sequence, selection or repetition control constructs.
3. Construct a PMM with steps reflecting these control constructs.
4. Add resource roles to the PMM steps that reflect the resource use in the project instance.

A simple visualization is to use the custom fields of the Planner program to record the “Kind of step” data mentioned above. Not only do the fields record the data, but the arrangement of the data in the Planner task and resource views makes patterns in the data obvious.

As an example: consider the project of Fig. 2. If we classify and group the tasks and resources of the project according to made-up classifications of “create page”, “revise page” and “get feedback” for project tasks, then Planner will present the task view of Fig. 4.

There are obvious patterns in the task classifications of Fig. 4: the “create page” and “revise page” tasks are grouped together, and each group is followed by a “get feedback” task. From this, we could guess control structures such as:

Name	Start	Finish	Work	Slack	Cost	PMM Step
▼ Create Web Pages	Nov 1	Nov 5	4d	0		Create web pages
Create CS101 web page	Nov 1	Nov 1	5h	4d 2h	0	Create page
Create CS102 web page	Nov 1	Nov 2	4h	3d 6h	0	Create page
Create CS201 web page	Nov 2	Nov 2	3h	3d 3h	0	Create page
Create CS202 web page	Nov 2	Nov 3	6h	3d 5h	0	Create page
Create CS301 web page	Nov 3	Nov 3	2h	3d 2h	0	Create page
Create CS302 web page	Nov 3	Nov 4	6h	3d 4h	0	Create page
Create CS303 web page	Nov 4	Nov 5	4h	3d 6h	0	Create page
Get feedback from academic staff	Nov 5	Nov 5	7h		0	Get feedback
▼ Revise Web Pages	Nov 5	Nov 10	2d 4h	0		Revise web pages
Revise CS101 web page	Nov 5	Nov 8	5h	5h	0	Revise page
Revise CS102 web page	Nov 8	Nov 8	2h	3h	0	Revise page
Revise CS201 web page	Nov 8	Nov 8	1h		0	Revise page
Revise CS202 web page	Nov 8	Nov 9	1h	5h	0	Revise page
Revise CS301 web page	Nov 9	Nov 9	2h	2h	0	Revise page
Revise CS302 web page	Nov 9	Nov 9	2h		0	Revise page
Revise CS303 web page	Nov 9	Nov 10	1h	5h	0	Revise page
Get feedback from academic staff	Nov 10	Nov 10	1d		0	Get feedback

Fig. 4. The task view of Planner, showing the task groupings and classifications with which we decorated the project instance

- While there are course web pages left to work on, work on the web pages.
- Change the course web pages until the academic representative is satisfied.
- If a course web page doesn’t exist for a course, then create it, otherwise, edit the existing page.

Finally, we could synthesise the control structures into a “Change Course Web Pages” PMM like this:

- Iterate the following steps until the academic representative is satisfied with the course web pages:
 - While there are course web pages left to work on:
 - * If a course web page doesn’t exist for a course, then the web master creates it, otherwise, they edit the existing page.

Having created a PMM, we can compare it to other PMMs and attempt to improve it. For example: a well-known strategy in IT projects is to run a pilot phase to elicit early feedback from the client. If we were to assume that the changes to the course web pages were all of a similar nature, then changing just one page and obtaining feedback for that page might reduce or remove the second round of web page changes we can see in the project instance of Fig. 2. The resultant PMM would be as follows:

1. The web master edits an existing web page to illustrate the kind of changes envisaged, and then
2. The web master obtains feedback from the academic representative about the changes, and then
3. The web master alters the scheme for changing the web pages, as appropriate, then
4. Iterate the following steps until the academic representative is satisfied with the course web pages:
 - While there are course web pages left to work on:
 - If a course web page doesn't exist for a course, then the web master creates it, otherwise, they edit the existing page.

8 Conclusions

Conventional PM promotes a static project plan, where all dimensions of the project; (tasks, resources and allocations), are frozen at the beginning of the project. This plan may be made up in some ad hoc way or produced from a template. The project is managed to follow the plan as closely as possible, because conventional PM has difficulty in coping with change in a project. However, both the formulation and management of a real project need specialist knowledge from the domain in which the project operates, to describe what tasks and resources are required to accomplish a goal and also to control the progress of the project.

PMMs formalize this specialist knowledge and form a reactive system that constrains the course of a project, according to the current state of the project. This is a more powerful and expressive concept than conventional PM, however, it results in a project that constantly changes its composition. For example: the operation of *selection* construct in a PMM results in one task being selected from several possibilities, and this is something that does not normally happen in conventional PM. PMMs such as XP have workarounds for this characteristic.

We have used data modelling to create a framework that relates PMMs to conventional PM. A simple Java program was described which implemented the framework and could create a “PMM-aware” project, given a PMM description and project goal.

The framework can also assist with producing visualizations of project data that could be used to synthesise PMMs to fit “non-PMM-aware” project instances. A simple visualization and PMM-creation heuristic was given as an example, and examination of the PMM so created revealed possible improvements to it.

Future work will centre around implementations of the framework with a variety of process description languages, the investigation of larger and more realistic applications of the PMM-based project generator, and the production of more visualizations of project data to, for example: compare and evaluate PMMs.

References

1. Burbridge, R.N.G.: Introduction. In Burbridge, R.N.G., ed.: *Perspectives on Project Management*. IEE Management of Technology Series 7. IEE, London (1988) xv–xxv
2. Standish: *Extreme Chaos*. In: <http://www.standishgroup.com>, The Standish Group (2001)
3. Standish: *The Chaos Chronicles*. In: <http://www.standishgroup.com>, The Standish Group (1994)
4. Lewis, J.: *Fundamentals of Project Management*. AMACOM, New York (1995)
5. Microsoft: *Microsoft Project 2003*. In: <http://www.microsoft.com/>, Microsoft Corporation (2003)
6. Flowers, S.: *Software Failure, Management Failure: Amazing Stories and Cautionary Tales*. Wiley, Chichester; New York (1996)
7. Collins, T., Bicknell, D.: *Crash, Ten Easy Ways to Avoid a Computer Disaster*. Simon and Schuster (1997)
8. Small, D.F.: *Ministerial Inquiry into INCIS*. In: http://www.justice.govt.nz/pubs/reports/2000/incis_rpt/index.html, Wellington, NZ Justice Department (2000)
9. Hult, R., Hallendal, M., del Castillo, A.: *Planner, a Project Management Application for the Gnome Desktop*. <http://www.imendio.org/projects/planner/> (2004)
10. Gray, C.F., Larson, E.W., eds.: *Project Management, The Managerial Process*. McGraw–Hill, New York (2000)
11. Fayolle, A., Scheller, M., Roberts, J.: *PyGantt, a Gantt Chart Generation Tool*. In: <http://www.logilab.org/pygantt>, Logilab <http://www.logilab.org/> (2000–3)
12. PMXML: *Project Management XML Schema*. In: <http://www.projectoffice.com/xml/>, Pacific Edge Software (2000)
13. Gantt, H.L.: *Organizing for Work*. George Allen & Unwin Ltd, London, UK (1919)
14. Schach, S.R., ed.: *Object–Oriented and Classical Software Engineering*. McGraw–Hill, New York (2002)
15. Beck, K.: *Extreme Programming Explained: Embrace Change*. 6th edn. XP series. Addison–Wesley (2000)
16. Royce, W.W.: *Managing the Development of Large Software Systems: Concepts and Techniques*. In: *IEEE WESTCON*, Los Angeles (1970) 1–9
17. BPML: *Business Process Markup Language*. In: <http://www.bpml.org/>, BPML Initiative (2000)
18. Boehm, C., Jacopini, G.: *Flow Diagrams, Turing Machines and Languages with only Two Formation Rules*. *Communications of the ACM* (1966) 366–371
19. Westfechtel, B.: *Models and Tools for Managing Development Processes*. Volume 1646 of *Lecture Notes in Computer Science*. Springer, Berlin, London (1999)
20. XML: *Extensible Markup Language: XML*. In: <http://www.w3.org/xml>. (1998)
21. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Design Patterns: Elements of Reusable Object–Oriented Software*. Addison–Wesley, Reading, Massachusetts (1995)
22. Dale, T.: *TPMM and PM/PMM framework*. In: <http://www.cosc.canterbury.ac.nz/tony.dale/msc/index.html>. (2005)
23. DOM: *Document Object Model Level 3 Core Specification*. In: <http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/>, The W3C Consortium (2004)
24. XSD: *XML Schema Definition*. In: <http://www.w3.org/2001/XMLSchema/>, The W3C Consortium (2001)