



## **Priority-Driven Processes**

***How to design processes for projects that  
use priority as a key management control***

*A whitepaper by*  
**Andy Carmichael, PhD, CEng, FBCS**  
*Vice President International Operations, Ivis Technologies*

### **Issue 2.0**

Ivis Technologies  
5110 N. 44th St  
Suite 200L  
Phoenix, Arizona 85018  
United States  
Tel: +1 (602) 343-8200  
Fax: +1 (602) 926-2348

[www.ivis.com](http://www.ivis.com)

# Priority-Driven Processes

## How to design processes for projects that use priority as a key management control

### Abstract

By defining the processes behind its common project types, an organization can improve productivity, quality and compliance with its best practices. However not all processes are created equal, and imposing rigid or inappropriate processes will be wasteful and counter-productive. Processes which schedule activities by evaluating which activity provides the greatest payback soonest are known as priority-driven processes. They use the priority of the deliverables as a key management control, along with the control of resourcing levels, release dates and project scope. This change from more traditional planning methods can improve risk management, efficiency and resource utilization and result in faster deliveries of the key requirements.

This paper, the second part of a two-part discussion, looks at applying the process improvement environment *xProcess* to projects that use priority-driven processes. It considers how to design processes that use such prioritization alongside the project patterns, task patterns, artifact templates and tailored quality checking procedures that *xProcess* supports. Its companion paper, *Planning By Priority*, focuses more especially on tracking and optimizing live projects with *xProcess*; it is also available from [www.ivis.com](http://www.ivis.com).

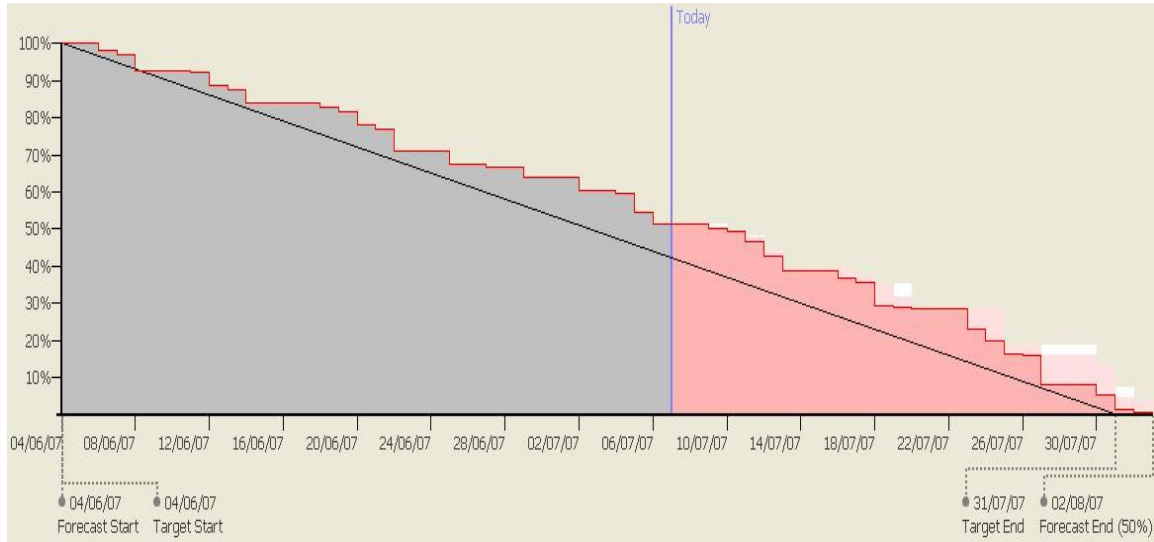
### Introduction

Prioritization is a key management control for scheduling project activities. In particular the ability to re-prioritize activities throughout a project is a vital strategy for ensuring both agility and successful outcomes. Often a successful outcome is considered delivering a project “on spec, on time and on budget”. While many would happily settle for that often elusive goal, there is actually a higher target to shoot for. Consider this feedback from research into corporate projects.

‘For executives, a “good” project was one that built the product that it set out to build. A “bad” project, according to executives, was one in which the final results ended up looking completely different to what they set out to build. Yet the market reaction to the project that had gone through continual change was much better than the project that had a design that was frozen in time. The way they thought about “good” and “bad” was completely upside down. The people who were overseeing projects assumed that the good projects were the ones that delivered to the spec. In fact, good projects are ones that deliver to the market.’ *Alan MacCormack, Harvard Business School (quoted by Tom Gilb [1])*

The challenge posed by the phrase “delivering to the market” is not inconsiderable. Those who achieve it do so by seeking continual feedback throughout the project lifecycle and continually adjusting priorities accordingly. This is why priority-driven processes have become so important in recent years.

A companion paper [2] considers how to “plan by priority” by examining an example project tasked with developing a retail web site. It shows how prioritizing tasks – and using automated scheduling of projects to produce visualizations such as forward-looking Burndown and Earned-Value charts (see Figure 1) – provides improved decision support for release scheduling and resource management.



**Figure 1. Tracking of Burndown or Earned-Value improves decision-support**

In this paper we look at the implied process behind such projects and show how a team can capture their process in *xProcess* from the plannable activities that they undertake. They can then improve this process as it is used on current and follow-on projects

The defined process need not just address task planning but also aspects such as quality control checks, artifact templates and management (for artifacts like requirements, issues, risks, design and user documentation), workflow (for notification of participants and integration with other systems in the environment such as accounting and tracker packages) and human resources concerns (such as the definitions of role and skill types). As organizations grow the libraries of processes behind their projects, they can share best practices, evaluate the effectiveness of process changes and optimize their standard approaches to a wide variety of project types. In the mean time, all the projects applying the processes are able to dynamically report status changes, target changes and scope changes achieving both improved agility, and demonstrable compliance and auditability.

To define a process in *xProcess* a number of process elements are available, such as:

- Task patterns
- Folder patterns
- Project patterns
- Artifact types
- Role types
- Gateway types
- Category types
- Expense types
- Actions

In the subsequent sections of the paper we will consider what these elements are and how they support process definition, and the planning and execution of projects following the process. There are many examples of defined processes for agile software development and the one we will model here is based on principles that they all share. By and large the process uses terminology from Feature-Driven Development (FDD) which is an approach developed by Jeff De Luca, Peter Coad and others [3, 4]. Equally users of methods such as XP [5], DSDM [6], Scrum [7], Evo [1] and UP [8] should find this material relevant and applicable to their processes too. Naturally terminology and templates differ, but these are the easiest parts of a process to modify.

The key common aspect of all these processes is that they are priority-driven, which means that the ordering of activities is determined by the incremental business value of the product they produce, rather than by grouping types of activity into a set order. Importantly tasks focus on one requirement at a time, and it is this switch to much smaller units of independently plannable activity that give these processes their agility and economic advantage. The “*features*” in FDD, “*user stories*” in XP, or “*use cases*” in UP, are scheduled according to their priority (based on risk-benefit), and thus agility in these methods arises from the ability to reprioritize these single requirements during the development process. In order to support this aspect of agile methods, we start by considering process at a finer level of granularity than with waterfall-style methods. We need to consider the process around a single requirement or feature – the Feature task pattern.

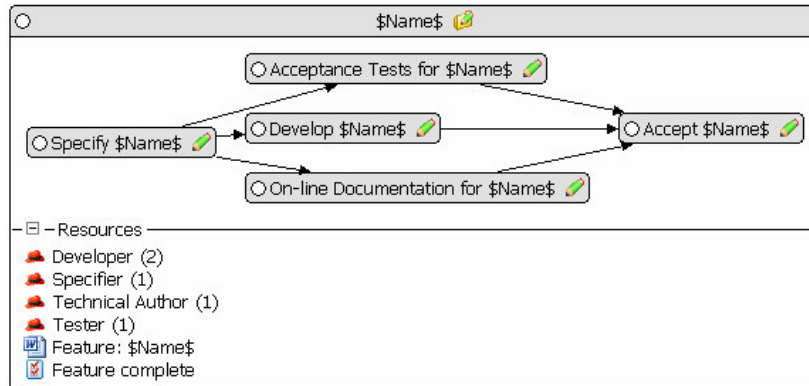
## A task pattern for a *Feature*



**Figure 2. A task pattern**

The simplest approach to modeling a feature task pattern is to use just a single task and attach to it the artifact templates that will be used in completing the task. Figure 2 shows such a task pattern. It has three attached artifacts: Word documents for Specification, Design and Test (they might equally be Wiki pages or files from some other tool). The pattern also defines the number and type of roles required to carry out the task; in this case up to two “Participants” (the most generic role type) and it may also define other aspects of the task such as the estimates of best-case, most-likely and worst-case effort required to complete the task. Since the pattern is used many times to make features in a real project, the pattern may be parameterized, and the values of the parameters, for example the “Name” of the feature, may be used for text-substitution in text fields and for other actions when instantiating the pattern. This simple pattern can be very effective in many cases.

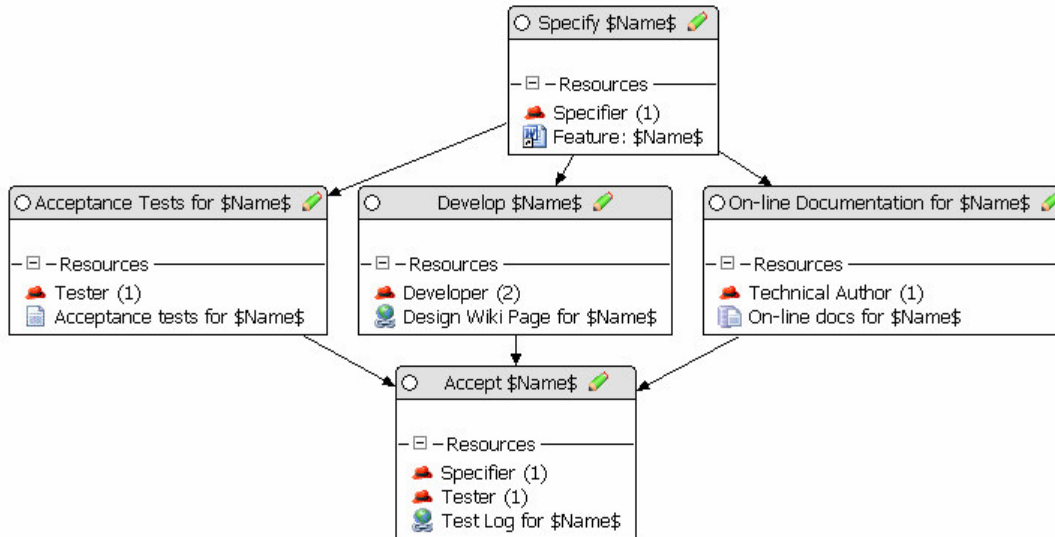
Typically however, the delivery of a single feature might involve collaboration by several people playing different roles, that each require completed work from someone else. In such a case it is worth considering a more complex task pattern. FDD itself [3] defines the sub-process for a single feature, breaking it down into “Design by feature” and “Build by feature”, each of which requires several roles that collaborate on the feature development.



**Figure 3. More complex task pattern for a Feature**

Figures 3 and 4 show yet another possible task pattern for a feature, where the task breakdown is more focused on the required artifacts to be produced by these roles.

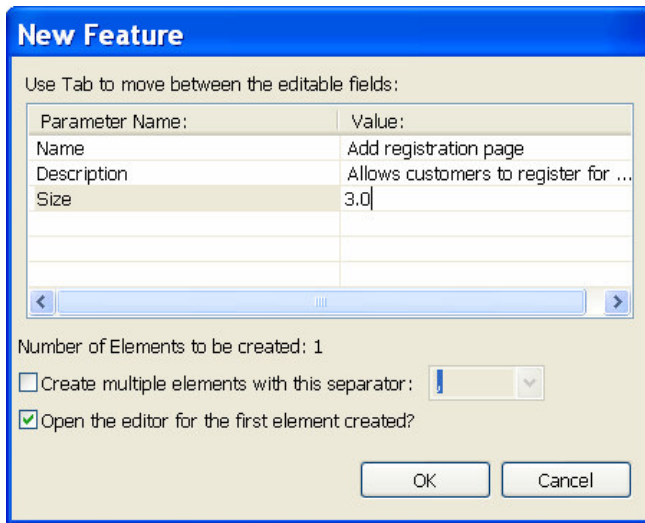
In this pattern we have defined five subtasks to be carried out on each feature – with dependency constraints to define the ordering of the subtasks – and four different role types: Specifier, Developer, Tester and Technical Author to ensure the auto-scheduler in *xProcess* assigns an appropriate person to each task. In practice one person may play several of these roles, for example being both a Specifier and Technical Author. Also some tasks may be carried out by more than one person: in this case we have defined the “Develop” task to require up to two developers



**Figure 4. “Go into” view of this pattern**

The pattern itself consists of “prototype” elements which can be defined in a similar way to tasks, folders and artifacts in a real project. These prototype elements are then cloned when the pattern is used.

Having defined a pattern in this way we can “instantiate” the pattern in a project. Figure 5 shows this being done and it reveals some other aspects of the pattern, particularly the parameters. Firstly notice that the name of the pattern “Feature” appears in the user interface



**Figure 5. Instantiating a *Feature* pattern**

to identify the action being taken. In other words the terminology of an organization’s own process is used in each case, rather than requiring managers and participants to learn a different terminology. This particular pattern has three parameters Name, Description and Size, which are defined by the process definer. The values provided by users of the pattern will tailor each pattern instance. In this example the parameter values will set the name and description fields as well as several other attributes related to the nominal size of the feature.

The process definer has a choice as to how parameter values will be used. If the parameter has a textual value, the text can be substituted in any text field in the prototype elements (by surrounding the parameter name between a pair of \$ characters). Alternatively an “Instantiation Action” may be used to modify the elements created from the pattern using parameter values. We can see from Figure 3 that in this case the *Name* parameter has been used to set the name of the tasks in the pattern by text substitution. The text string \$Name\$ appears in several places showing where the actual name provided by the user of the pattern will appear. Similarly, though not shown on the diagram, the description field of the parent task in the pattern contains \$Description\$ in order to set this field.

The *Size* parameter (which is provided as a way to provide a nominal comparison of feature sizes) is used in an “Instantiation Action” of the pattern called *Set estimates*. An Instantiation Action in *xProcess* is called whenever its pattern is used and tailors the pattern to its usage context, for example setting dates relative to the current date or using parameter values. The *Set estimates* Instantiation Action sets the *size* attribute of all the tasks in the pattern as well as the *best*, *mostLikely* and *worst* attributes, which represent the three-point estimates of required effort.

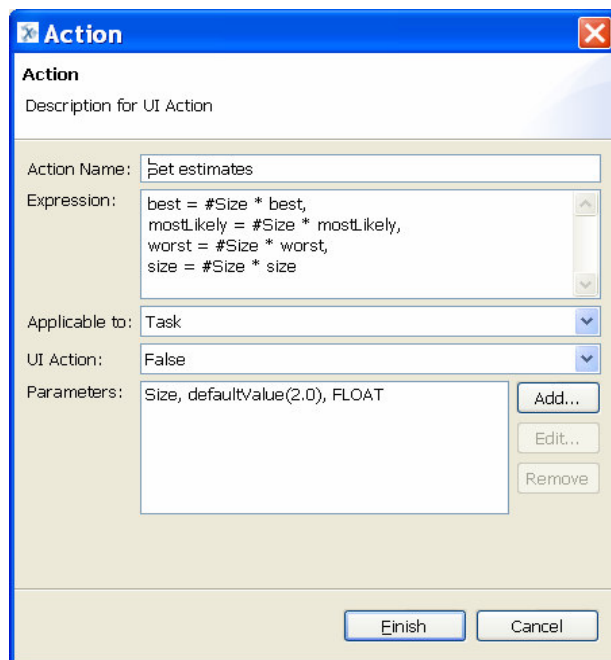
Actions in *xProcess* are written in a general scripting/rules language called OGNL [9] which gives very general access to the underlying model in a clean and understandable syntax. This particular action is shown in the dialog box in Figure 6<sup>1</sup>.

<sup>1</sup> The OGNL code shown in Figure 6 takes the existing values for *best*, *mostLikely*, *worst* and *size*, and scales them by the value of *size* from the parameter, #Size.

Instantiation Actions like this may be applied to all or a selection of the prototype elements in the pattern, making this a very flexible mechanism for defining appropriate patterns in a wide variety of processes. Pre-defined actions (including this one) are provided “out of the box” in *xProcess* for most common uses of Actions. As well as for Instantiation Actions, OGNL is used in a number of other places in *xProcess*, including:

- Setting default values for parameters (“Default Value Getters”)
- Providing actions which may be invoked directly by users (“User Actions”)
- Defining actions to be triggered by workflow events
- Defining States (from which state transition events are derived to trigger workflow)
- Defining process-specific rules.
- Configuring pages for the *xProcess* Web Client.

The Feature pattern example discussed in this section has shown that patterns of different complexity may be modeled from prototype objects that are cloned when the pattern is used. Since all aspects of the prototypes are cloned on instantiation, even aspects such as typical expenses associated with tasks can be included for financial planning. We have also seen that parameters, text substitution and Instantiation Actions may modify the objects that are created from patterns. In the next sections we consider firstly project patterns, and then patterns for planning boundaries within processes, the main component of which is the Task Folder construct.

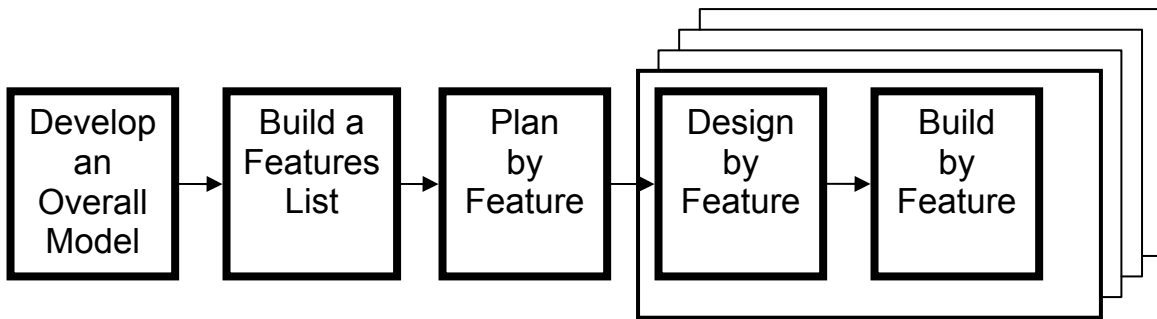


**Figure 6. Example of an Action**

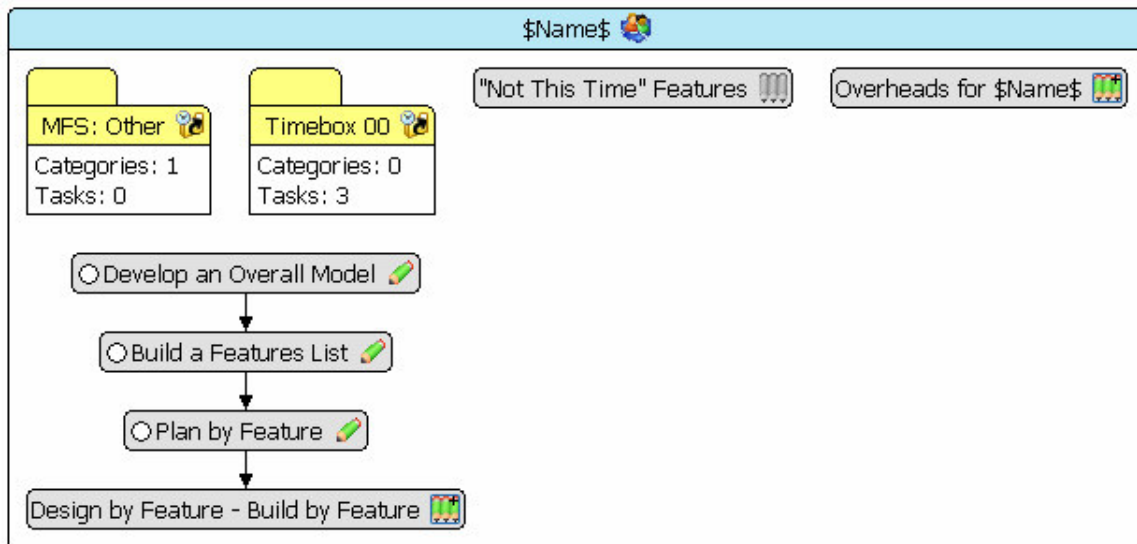
## Project patterns

Generally it is important to add at least one Project pattern to a new process since this allows project managers to choose the process at the same time as creating the project itself. It also means that the always-repeated activities of the projects – particularly startup, closedown and overheads tasks – are included from the start of planning.

Figure 7 shows the process diagram most commonly used to describe FDD [4]. It shows three initial activities followed by a set of iterated activities which are repeated for every feature. Figure 8 shows a possible project pattern for FDD in the *xProcess* pattern editor. As well as the four top-level FDD processes, this diagram also shows: the first timebox folder (representing the first timed iteration of the project); a “Major Feature Set” folder (Major Feature Sets group features by business area); and two other parent tasks for features not expected to be scheduled within the scope of the project, and for overheads. Subsequent iterations will be created from the timebox pattern as they are planned.



**Figure 7. The FDD Process**



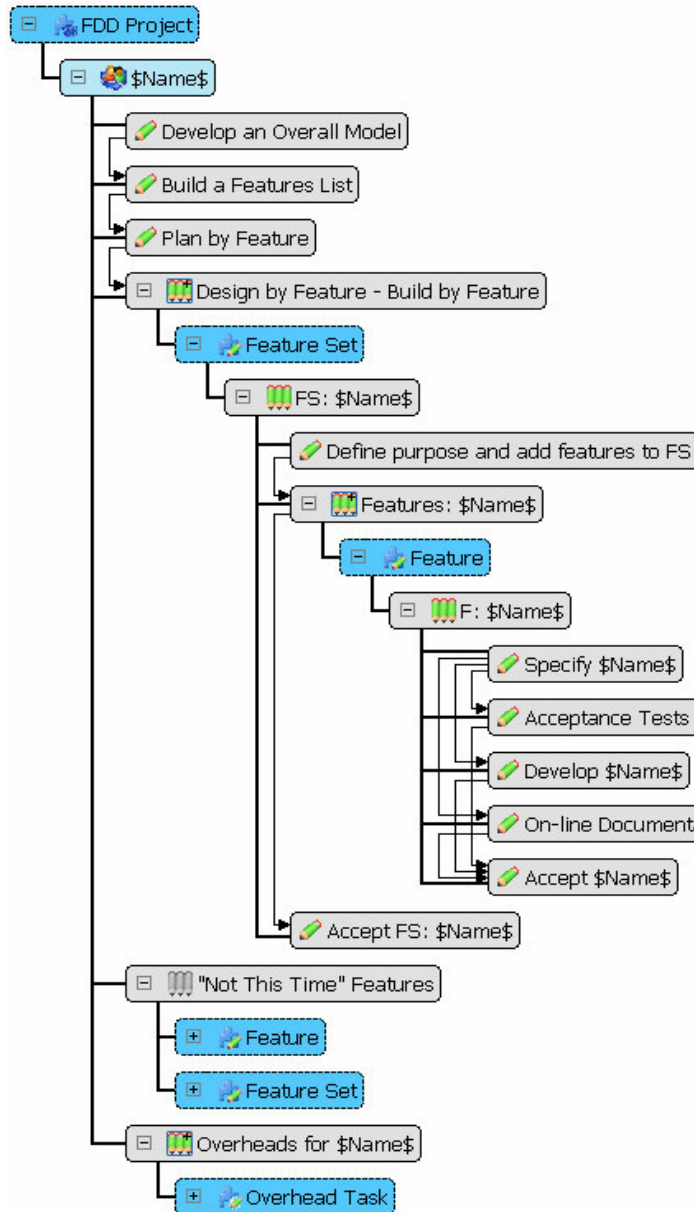
**Figure 8. FDD Project Pattern**

Further detail of the project pattern can be seen in its hierarchy view shown in Figure 9. This shows the use of a number of composite tasks that consist of a collection or selection of pattern instances – for example the “Design by Feature - Build by Feature” task consists of a number of instances of the “Feature Set pattern; the “Not This Time” task may contain instances of either the Feature or Feature Set patterns. We can also see the structure of these patterns in this view.

## Phases, Timeboxes and Prioritization

Most processes for projects divide the work into divisions such as phases, milestones, timeboxes and releases. These planning boundaries are important in providing key monitoring points to ensure the work is on schedule and delivering value. In *xProcess* the main ways to model such divisions are with Task Folders, Parent or Composite Tasks, and Categories.

The tasks in *xProcess* form a strict hierarchy from Portfolios of Projects, through Projects, Parent Tasks down to leaf-level tasks. Any leaf-level task can become a Parent Task by further decomposing it into subtasks. Similarly child tasks may be rolled up into



**Figure 9. Pattern Hierarchy View**

development activity (as in a Waterfall-style decomposition into say, requirements analysis, design, implementation and integration). In any case whichever decomposition form is chosen as the primary dimension of a process, there will be other groupings of tasks that are significant, and whatever these groupings are, it is likely that you will model them in *xProcess* with Task Folders, or patterns which contain Task Folders.

Task Folders are simple structures that contain “shortcuts” (or references) to tasks and target start and end dates. Optionally they may contain information about the relative pri-

larger tasks, each of which may require multiple role types, and/or multiple people to perform each role type. For example if the child tasks were removed from the task pattern shown in Figure 3, the remaining leaf-level task would still be scheduled to two Developers, one Specifier, one Technical Author and one Tester<sup>2</sup>. Furthermore detailed specification of parent tasks allows a choice of “top-down” or “bottom-up” planning, depending on whether all or only some of the subtasks have yet been identified.

The hierarchy is an important division of work and it is of course often employed in traditional planning methods, for example in Work Breakdown Structures. For priority driven methods to be effective however the hierarchy must be used with care. If the wrong dimension is used for the decomposition it becomes hard to see progress at a high level, and hard to prioritize one requirement against another. This is why FDD uses a decomposition structure (Major Feature Set, Feature Set, Feature) which corresponds to the requirements structure – the “user-appreciated deliverables” – rather than into the types of de-

<sup>2</sup> On the other hand, without the child tasks and their dependencies, certain constraints would be lost from the plan, possibly resulting in scheduling work before it could be completed.

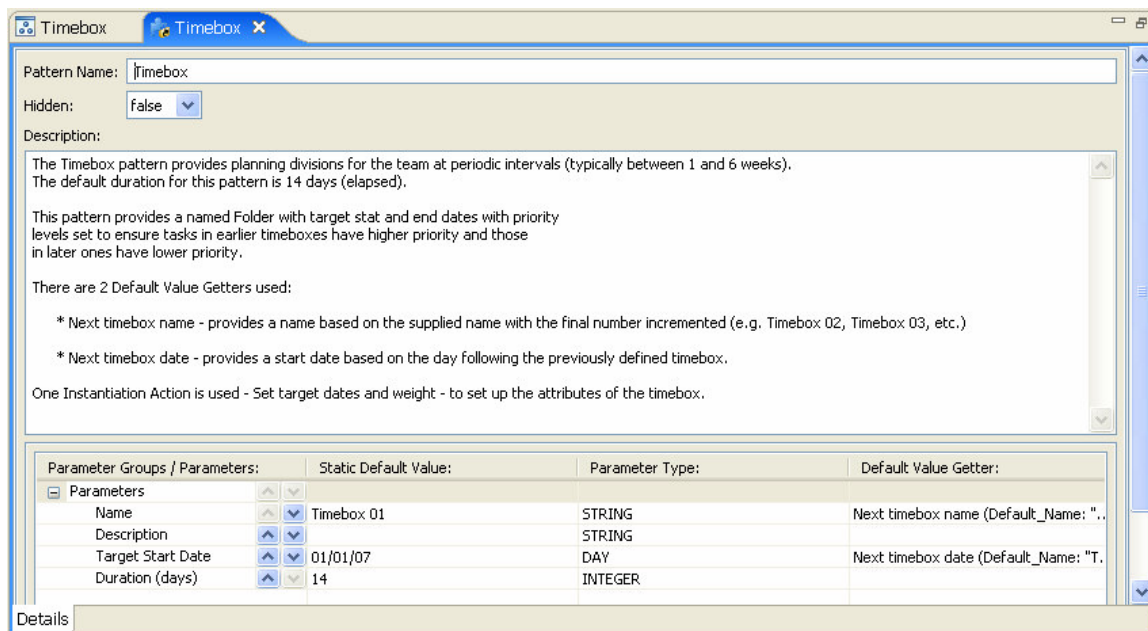
ority of the tasks in the folder and the relative priority of this folder, as opposed to other folders or parent tasks in the Project.

Task Folders are not part of the task hierarchy in *xProcess*. Tasks may therefore be contained in many different Folders for different purposes. The tasks may appear there as part of the *explicit* membership of the Folder, (because the particular task, or its parent / ancestor has been added to the folder), or because of *implicit* membership, for example if the membership of the folder is defined to include a Category into which this task falls. There can be many different reasons for defining Folders and usually it will involve the scope (or membership) of the Folder, the target dates for the folder (when its tasks should begin and end) and the relative priorities of the tasks in the Folder.

In the FDD-based process we are considering here, Task Folders are used to model the following elements:

- Timeboxes (periodic review structure for the development team)
- Releases (periodic release of functionality to users)
- Major Feature Sets (groupings of Feature Sets by business area).

Let's take a closer look at the Timebox pattern which is shown in Figure 10. The parameters and actions defined here provide a way for the project manager, the probable user of this pattern, to set these periodic team reviews on the dates and at the frequency that makes sense for his team.



**Figure 10. Details of Timebox Pattern shown in the Pattern Editor**

Other processes may wish to also use Task Folders for “phases” where this is preferred to using a Parent Task. For example processes may contain phases such as Feasibility Study,

the “Hot house” prototyping phase<sup>3</sup>, or the Deployment phase. It is even possible to ensure that different subtasks from patterns are prioritized within the different phases by defining the contents of the phases by Category and categorizing the subtasks of the patterns accordingly.

Of the three FDD patterns mentioned above the Major Feature Set pattern is perhaps unusual. One could have used Parent Tasks instead of folders to model this aspect of the process and it certainly could not be called incorrect to do so. The reason that instead Task Folders were used in this case was because this was found to give project managers greater flexibility. It also means that all the top level tasks in the resulting projects have meaningful start and end dates. Feature Sets are groups of Features which will probably be delivered together within one release. Major Feature Sets are more groupings into logical areas and are therefore likely to go across multiple release boundaries.

## Tailoring the documents used in the process

*xProcess* has the capability not only for capturing the tasks within a process but the artifacts that are created, modified or qualified as those tasks are executed. Patterns may include templates for standard document types so that when the pattern is instantiated, the artifact itself is created (including text substitution of parameter values). The artifact can then be accessed and edited by navigating from the task. Artifacts may also be *referenced* from patterns (as opposed to *included*), in which case a reference to the same artifact is copied to the new instance of the patterns rather than making a new artifact. Both these conditions are useful in different circumstances.

Artifacts may be used for many different purposes within software engineering processes. For example:

- Requirements
- Design Specs
- Object / Data models
- Test specs
- Test results
- Issues
- Risks
- User/Help documentation
- Standard meeting minutes

Artifacts can be cross-referenced, linked to tasks, browsed, versioned, built from templates, and they may be edited with user-installed tools. References to artifacts may be seen in several of the patterns that we have already reviewed – for example in Figures 2, 3 and 4.

The close linking of artifacts with the activities that create, modify or qualify the artifacts is an important aspect of most modern project management processes. *xProcess* allows this aspect to be well supported, and it is an essential consideration for process engineers configuring their process.

---

<sup>3</sup> See [10] for a process which uses this technique

## Tailored quality checking procedures

*xProcess* provides an execution environment for projects which is based on full versioning of the process, planning, time recording and artifact information. This means that information is available to process auditors seeking to improve the performance of projects, and comparisons can be made between closely aligned methods to see the effectiveness of process changes. A key input to understanding the effectiveness of processes, and diagnosing where problems may have arisen after the event, is information about how closely required elements of the process have been followed. *xProcess* provides a “Gateway” mechanism precisely to address this issue and to act as an *aide memoire* for project participants applying the process.

These quality-check definitions are simple to define and use. An example Gateway definition is shown in Figure 11. The Gateway consists of a series of questions, each with a selection of possible answers and an indication of whether each answer is a “pass” or “fail”. Failed answers to gateways do not prevent project participants from proceeding, but they do allow alerts or follow on actions to be raised and importantly they can be used in auditing and improving processes after the event.

Gateway Name:			
Critical Task Checklist			
Description:			
This gateway should be attached to tasks (or patterns for tasks) that required additional auditable checking			
No.:	Questions:	Answers For: Question 1:	Pass/Fail:
1	Has the task been completed in compliance with applicable standards and procedures?	yes	true
2	Have tests been developed for all required aspects and do the tests pass?	no	false
3	Are any follow-up actions required?		

**Figure 11. Defining a Type of Gate-**

Figure 11 shows this same Gateway in use by a project participant, who is closing a controlled task using, in this case the web client for *xProcess*.

The possible answers of the Gateway appear as *radio-button* multiple choices. Any “fail” responses require the user to enter the rationale for proceeding while in this state. These responses can be used to plan follow up action or compare outcomes from different scenarios. In most cases gateways are used as process guidance for participants on projects. However the gateways highlight to participants that they have responsibility for deviation from standard approaches and must can record the rationale behind such a decision.

This helps to ensure good compliance with processes, and will highlight where processes are not being followed, whether that is the fault of the process or its application.

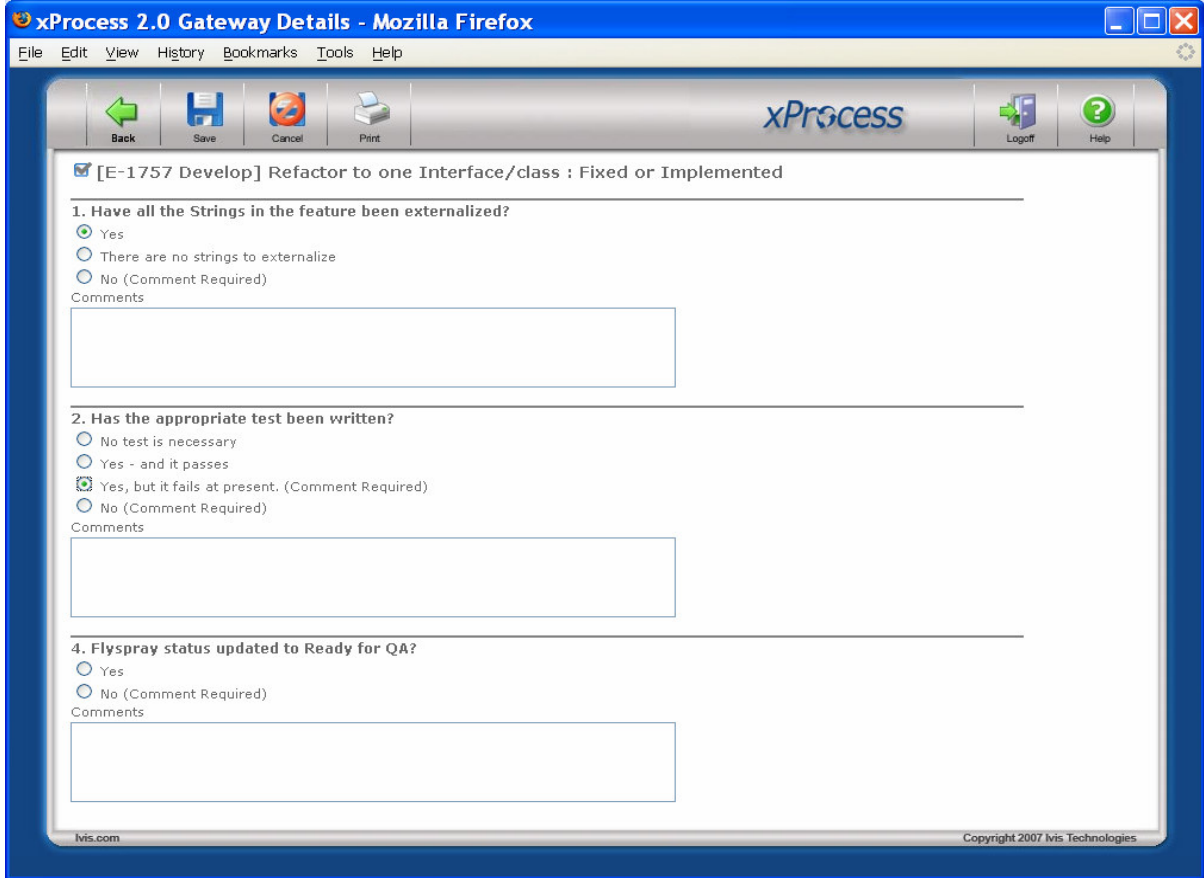


Figure 12. Completing the Gateway for a Task

## Conclusions

To gain control of the processes behind your projects, it is essential to be able to capture them and monitor how changes to them – or indeed deviations from them – affect productivity, agility and quality. An environment such as *xProcess* not only addresses the issue of graphically modeling the processes, it also provides the execution environment for projects applying them. This paper has shown how to begin to define a priority-driven process in *xProcess*, and how a project instantiated from the patterns of the process works effectively. Examples of the use of Task Patterns, Project Patterns Folder Patterns, Artifacts and Gateways have shown mechanisms for supporting methods like FDD, which combine agile management with effective requirements capture, modeling and planning

This paper is the third in a series on Dynamic Process Management and *xProcess* being published by Ivis Technologies (see also [2, 11]). Other papers in this series, as well as complementary materials and help with defining *xProcess* processes are available from [www.ivis.com](http://www.ivis.com). You can also provide feedback on issues raised in this paper at [xprocess.blogspot.com](http://xprocess.blogspot.com).

## References

1. Tom Gilb (2005) *Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*, Elsevier Butterworth-Heinemann, Oxford.
2. Andy Carmichael (2007). *Planning By Priority: How to prioritize project tasks to maximize delivered business benefits*, Issue 2.0, Dynamic Process Management with xProcess Series, Ivis Technologies LLC.
3. Peter Coad, E. Lefebvre, and J. De Luca. (1999). *Java Modeling in Color*. Prentice-Hall, Englewood Cliffs, NJ.
4. Stephen Palmer and Mac Felsing. (2002). *A Practical Guide to Feature-Driven Development*, Prentice-Hall, Englewood Cliffs, NJ.
5. Kent Beck and Martin Fowler (2001). *Planning Extreme Programming*. Addison-Wesley, Reading, MA.
6. Jennifer Stapleton (1997). *DSDM*. Addison-Wesley, Reading, MA.
7. Ken Schwaber and Mike Beedle (2002). *Agile Software Development with Scrum*. Prentice-Hall, Upper Saddle River, NJ.
8. Per Kroll and Philippe Kruchten (2003). *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*. Addison-Wesley, Reading, MA.
9. Drew Davidson (2004). *OGNL Language Guide*, OGNL Technology, Inc.
10. Andy Carmichael and Ian Evans (2006). *Building Agility into a CMMI-oriented culture – the BT experience*, European Systems and Software Engineering Process Group Conference, London.
11. Andy Carmichael (2006). *Managing Processes with Agility and Transparency*, Issue 2.0, Dynamic Process Management with xProcess Series Ivis Technologies LLC.

## **About Ivis Technologies**

Ivis Technologies is a leading provider of software solutions to enable organizations to improve project planning and execution, while simultaneously improving their processes. Organizations that optimize processes and make the best use of resources, benefit with increased efficiency, quality and productivity. xProcess helps organizations become more efficient, productive and agile. It is an environment that brings together process management with project planning, forecasting and execution. xProcess helps skilled professionals collaborate to fulfill complex goals and to optimize plans, improve processes and manage resources.

Founded in 2002, Ivis Technologies' products have been adopted by leading organizations in several industries. Ivis Technologies is headquartered in Phoenix, AZ with additional offices in Southampton, UK.

### **Ivis World Headquarters**

5110 N. 44th St  
Suite 200L  
Phoenix, Arizona 85018  
United States  
Tel: +1 (602) 343-8200  
[sales@ivis.com](mailto:sales@ivis.com)  
[www.ivis.com](http://www.ivis.com)

### **Ivis International Software (UK)**

Westwood Centre  
Nutwood Way  
Totton  
Southampton SO40 3SZ, UK  
Tel: +44 (0)23 8087 5899  
[eurosales@ivis.com](mailto:eurosales@ivis.com)  
[www.ivis.com](http://www.ivis.com)