A pattern in the making: Bridging the SCM maturity gap

Jonas Bovin, ALOC A/S

Abstract

In today's software development environment, the one thing we can be sure of is that fewer and fewer people are actual software developers. Today, we also need graphical designers, a number of domain experts, usability experts, testers and so on. We expect them to deliver work products to our SCM solution like the developers – but can we really expect them to behave like software developers or do we need to bridge their SCM maturity gap? This paper will describe, both in specific and general terms, one way to spot the gap as well as bridging it.

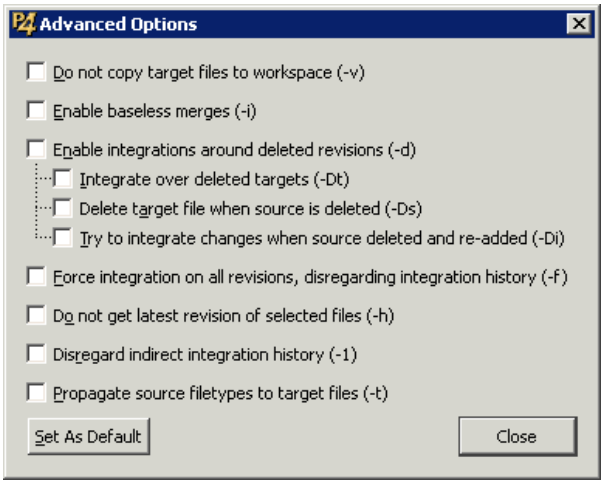
Background

In ALOC, as in many other software companies today, the number of people involved in software development who are not software developers is higher than ever. Designers, domain experts, usability experts, testers etc. are forced to work with the same SCM tools as the developers, in order for them to deliver. In ALOC we use Perforce to handle revisions on source code and other artifacts, as well as versioning and releases of our deliverables.

Historically, the branch, merge and integrate functions have been managed by one SCM master. In spring 2009, we introduced a new branching strategy based upon Patterns from "Software Configuration Patterns – Effective Teamwork, Practical Integration"[i]. With this new strategy we wanted to support Continuous Integration and increase the possibilities to go Agile. But as we created the branching strategy, it became apparent that the existing process needed changing as well. The SCM master would clearly be a bottleneck, but it would also be costly having two persons for each integration task – the SCM master and the person who changed the code. The number of integrations was believed to increase as daily deliveries were integrated to other branches and fixes had to be fetched.

We considered educating the non-software developers in the use of P4V, but came to the conclusion that their vocabulary and their understanding of SCM was lacking in a way that it could not be explained to them without extensive training. This task was unfeasible.

P4V is excellent if you are familiar with version control systems and their usage and have an understanding of the associated lingo on a general level. However, when looking at the 'Advanced Options' dialog, it is apparent that it is in fact still based upon a low level command line tool. The wonders of graphical user interface are of little help to inexperienced users when used in this way.



On the next couple of pages, we will explore how we solved this by making an application that implement the SCM domain specific terms but abstracts the underlying command line details. We will also describe the solution in

general terms and in the form of a pattern

What is a Pattern?

A pattern is a proven good solution to a problem in a context. An anti-pattern is a proven bad solution to a problem in a context. Patterns have their roots in architecture. The architect Christopher Alexander (1936-) introduced the notion of a pattern language which he explains as:
"The elements of this language are entities called patterns. Each pattern describes a problem that occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice."
In the field of software development the 'Gang of four'[ii] book has earned great recognition. In 2002 "Software Configuration Patterns – Effective Teamwork, Practical Integration" was published which contained a number of SCM-patterns, including branching patterns and policy patterns, which we use extensively in ALOC.

Most of the patterns that Alexander wrote are in the form:

```
IF   you find yourself in CONTEXT
     for example EXAMPLES,
     with PROBLEM,
     entailing FORCES
THEN for some REASONS,
     apply DESIGN FORM AND/OR RULE
     to construct SOLUTION
     leading to NEW CONTEXT and OTHER PATTERNS[iii]
```

In the following we will try to use Alexander's form to describe our pattern. We will first present the plug-in specific pattern in italic fonts and then the abstracted general pattern with a bold heading. The specific pattern will then serve as an example of each element in the pattern.

Writing the pattern

From the "Background" and the "What is a Pattern" introduction we are now able to define the first part of the pattern:

CONTEXT: Development with several different backgrounds (economic, financial, software) not being able to use SCM processes as they have very different knowledge of SCM.

-could in general terms be described as:

CONTEXT: A group of people are to work towards a common goal where the tasks require the application of processes from a domain, but without having the necessary insight into the domain, resulting in non-achievement of the goals.

EXAMPLE: The specific context could apply as an example.

PROBLEM: There was a low level of confidence in developer's own SCM work, leading to poor quality. In the end, an SCM master and a developer had to do the SCM work together. This was very inefficient and only increased work quality somewhat.

PROBLEM: The significant gap between actual and required knowledge to use available tools leads to inefficient and error prone work processes, low level of confidence and poor work quality. The lack of common level of domain knowledge leads to a lack of a common problem language.

FORCES: Heavy workload meant little time for SCM education. Some sub-projects have high risk which could be handled by SCM e.g. by isolating work on branches. SCM not perceived as a possible remedy, but solely as a revision control repository. The team was aware of many of the symptoms and was actively seeking help.

FORCES: Gap in domain knowledge cannot be filled with the time and resources available. The lack of domain insight leads to the inability to correctly use the tool of the domain. There is an awareness of problems and willingness to seek help.

Bridging the SCM maturity gap

Our first step was to define a common vocabulary. What must be named in order to communicate clearly about required actions? Beyond the known terms like main, release, submit, branch etc., we introduced some new terms including

- deliver (from child branch to parent)
- catch-up (from parent to child)
- baseless (from an early release to a later release)
- the relevant pattern names

It was described as a best practice that could be adapted to a given situation. We taught the basics of the different branching patterns that were in our proposed strategy and also the 'Branch Policy' patterns which we were going to use for every branch.

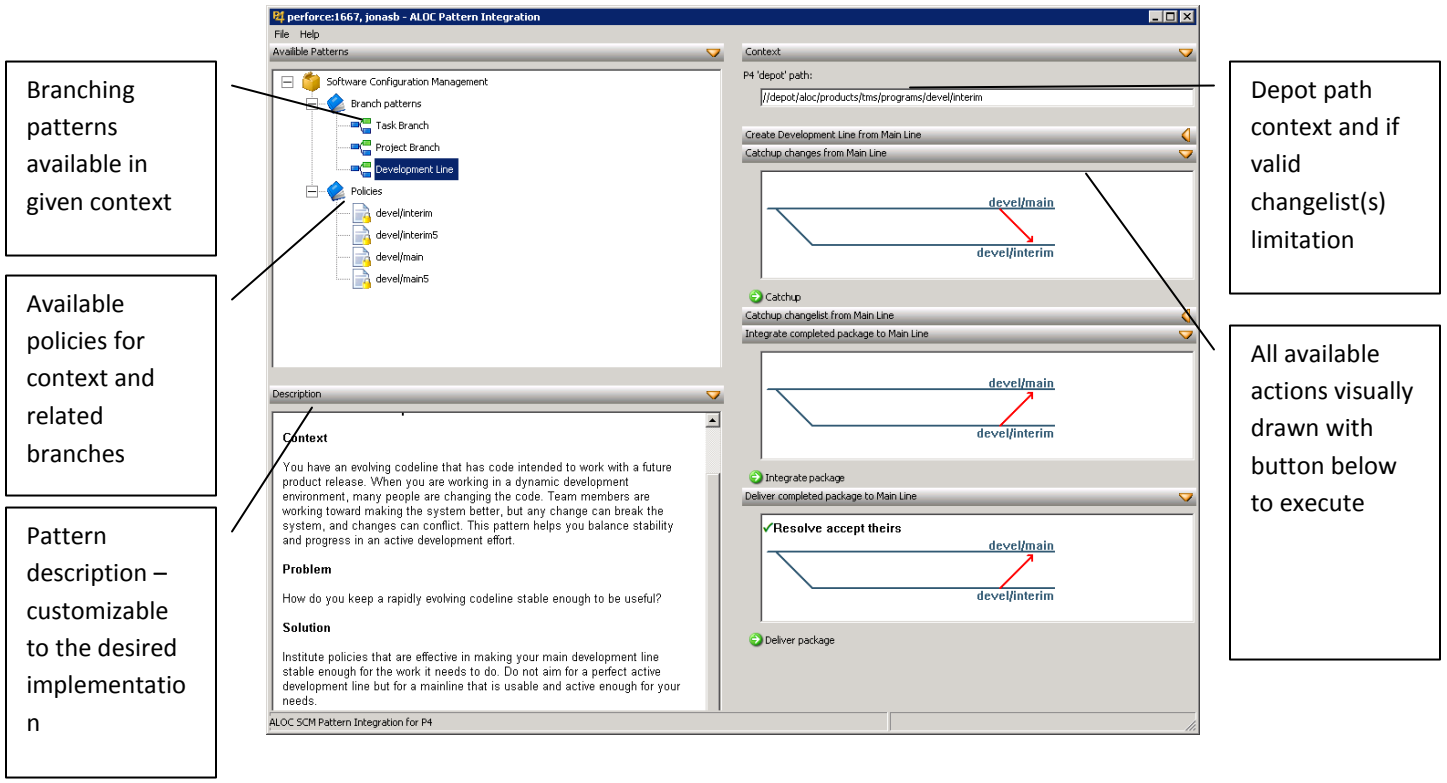
It seemed that after only a few sessions where we explored the terms and their implications, the team were able to understand and communicate the steps involved in the branching strategy – without ever having created a branch mapping or even seen the P4V integration window.

While we educated the team I developed the first version of the Pattern Plug-in. It worked by using Custom Tools in either P4V or P4Win and adds to P4 context of path, changelist or job.

Flexibility in its use was paramount so we decided that everything should be specified in xml-documents. When creating a new pattern, simply create a new xml-document describing the pattern. Also, it was important to us that no wrong choices were available to the user of the Pattern Plug-in. Based on naming conventions of branches as well as branch mapping settings, we are able to validate for a number of prerequisites and only show patterns and actions that are permitted according to context and state.

We incorporated a 'description' field in the GUI for a number of reasons. We wanted to be able to have the entire pattern text from context to solution available the moment it was selected, and we also wanted to be able to show a 'pretty print' version of the policies. The policies are also specified in XML, but there is a viewing and editing GUI within the pattern plug-in. The content of the policies are also enforced via the GUI – if anyone created a private branch, the policy can ensure them privacy. The GUI always describes the action as well as showing a simplified branching drawing that visualizes the action.

The result of an action in the Pattern Plug-in is always a new pending changelist – which can be reviewed before submission. This, along with the fact that only permitted actions are shown, ensures that users are confident that they are not going to get into trouble whatever they do.



The Pattern taking shape

From the solution we can now extract the rest of the elements in our pattern.

REASONS: We felt the patterns could help the team overcome their problems. We were able to extend Perforce tools. We had an idea of how to make SCM patterns available to the team through simple abstractions based upon context, state and policies.

REASONS: Symptoms and problems can be solved by using elements from the domain. There exists a way to create a simplification/abstraction of elements which can be more easily understood.

FORM: We were able to use current tools as a platform, our abstractions could be implemented in the form of context sensitive dialogs on path, changelist and job.

FORM: The abstractions can be embedded or integrated into existing context/platform.

SOLUTION: We supply context sensitive tools that free the user from the underlying details. The tool supplies the necessary details to the underlying implementation based on the current context in which we have embedded relevant metadata. The SCM patterns should be understood by the team well enough to enable team members to confidently describe their projects' SCM goals and problems.

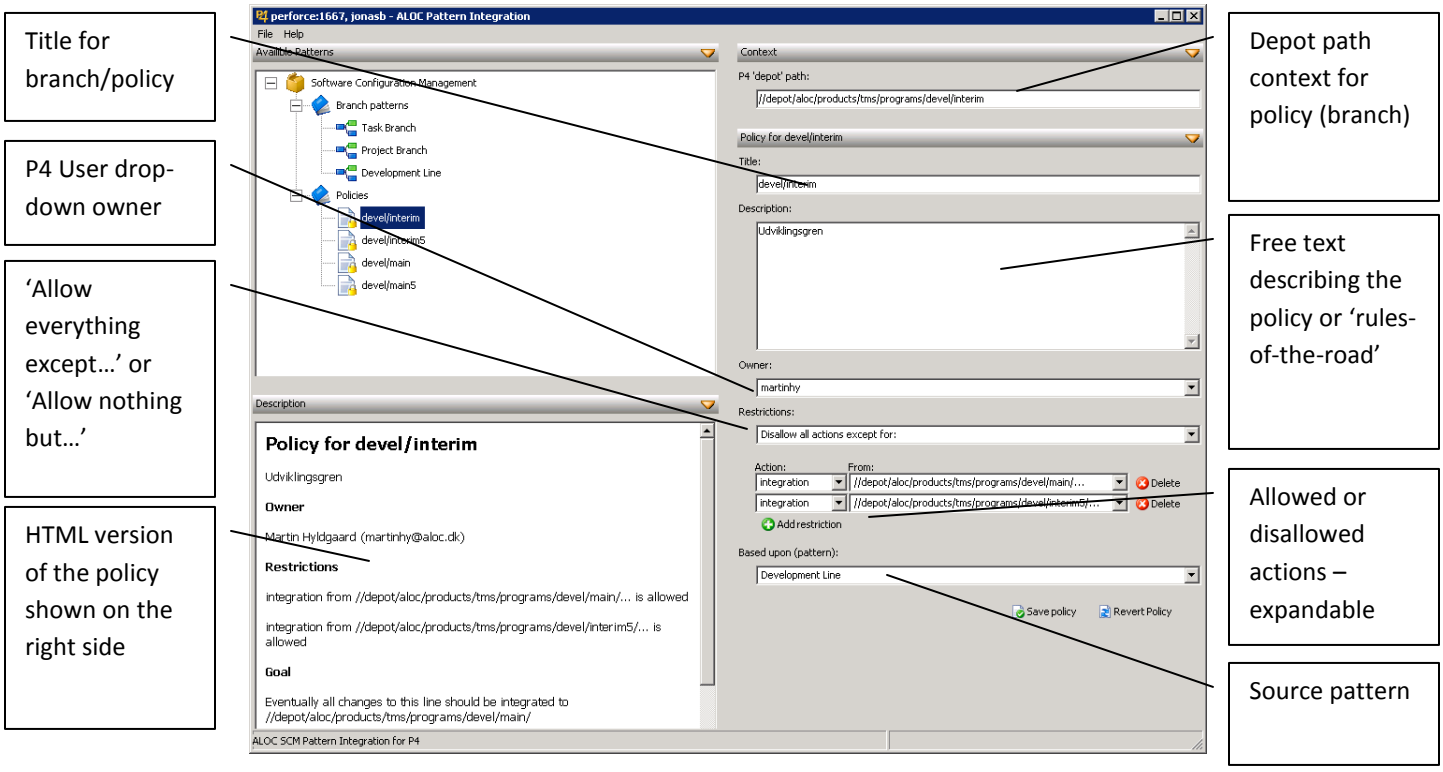
SOLUTION: The abstractions create a language which can be used with confidence to describe problems and goals. The abstraction is incorporated in an interface that allows the user to focus on parameters relevant to the abstraction and instead, the interface uses the abstraction supplied parameters and context to supply the necessary details to the underlying implementation.

NEW CONTEXT: Users benefit from SCM with minimal effort. The team eagerly explores more areas to improve, like automated build and test. When their needs change they can describe what they need to change. None of them knows what a branch mapping is, and only one of them has used the integration dialog from P4V.

NEW CONTEXT: A group of people are to work towards a common goal where the tasks require the application of processes from a domain, the users have available a set of abstracts to apply the necessary set of processes to achieve their goals.

The Pattern Plug-in

The pattern plug-in is, in essence, an abstraction from the command line interface based on the principles of SCM patterns instead of the technical implementation of Perforce. This approach is not a new one, and in true pattern fashion we display: ”an aggressive disregard for originality”[iv]. However, as we suggest in the title “A pattern in the making” this is by no means a full-blown pattern. Merely an approach that worked for us written in pattern style, but one could hope that some time in the future, a pattern similar to this one, would be utilized by software companies; realizing that one size does indeed not fit all. There will be a market for different types of user interfaces all written to solve the same functions but for people with different backgrounds.



Lessons Learned

The experiences we can draw from the first year in service is astounding. The heterogeneous group which previously needed a branch master for all branch creations and integrations has been transformed into a confident team which leverages on effective tools. Everyone in the group can do these actions in their sleep – and the branch master has even been moved to a different department. The total education amounts to about 4 hours split up in 3 session and they have

hardly required any support since. If the traces in the submitted changelists did not reveal the generated description, it is hard to believe that the department does not perform a single P4V integration – every single one is done through Pattern Plug-in. They only do the resolve in P4V – and they are the right people to do it.

The future of Pattern Plug-in

Although we have described our solution in a general way as a pattern – there are vast possibilities for development of the Pattern Plug-in. For one, we will have a pre-submit trigger on the server to check for adherence for the policy attached to the branch. The current solution only validates changes made via the GUI. In our current situation this is not an issue, but as it is adopted by the entire organization, it would ease the job of the Perforce Manager. E.g. not setting rights for each branch as this can be effectuated by branch creator.

Another aspect of the Pattern Plug-in we are keen to explore is the handling of processes. Typically, the changes created on a given branch are intended for a different branch – this is true for all development branches. For release branches the goal might be different. The introduction of a workflow in the product would really save time whether the branching is staged lines or development lines and release lines. Each submission will trigger a workflow. The workflow could consist of e.g.

- automated build
- automated test

Should the steps complete without error, then and only then, the change is added to an auto propagation queue, which in turn will integrate changes to the related branch and submit if there are no conflicts. If conflicts are found by the resolve action the result would be an entry in the user's to-do-list asking him to resolve the conflicts. A question that pops up is: Can the 'to-do' list in the new P4V Dashboard be programmatically manipulated? If so this would be the ideal location for a conflicting resolve.

When the process ends, the change should reside on an end line from where it can be released or whatever is needed. It would certainly keep the branch divergence to a minimum.

The original pattern plug-in is written in Python, which at the time fitted well with our other plug-ins. What we see for the future is a plug-in for Eclipse written in Java using P4 WSAD. I could see an Eclipse RCP version of P4V either as a standalone product or as elements in the developer workbench. An RCP edition will give much wider possibilities to integrate solutions like the Pattern plug-in. Eclipse supports a pluggable architecture in a way I do not see for the P4V. Much of the functionality of P4V is already in Eclipse – so I am eagerly awaiting the eclipse version where I can develop plug-ins that completely integrate into the GUI.

---

[i] Stephen P. Bercuzk with Brad Appleton "Software Configuration Patterns – Effective Teamwork, Practical Integration" (2002)

[ii] Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides "Design Patterns: Elements of Reusable Object-Oriented Software" (1994)

[iii] Patterns-Discussion FAQ: http://gee.cs.oswego.edu/dl/pd-FAQ/pd-FAQ.html

[iv] Pattern Value System: http://c2.com/cgi-bin/wiki?PatternValueSystem