Pattern-Oriented Approach to Software Process Evolution

(Submitted to IWPSE99)

Hajimu Iida (iida@itc.aist-nara.ac.jp)
Information Technology Center, Nara Institute of Science and Technology
8916-5 Takayama-cho, Ikoma-shi, Nara, 630-0101 Japan

Abstract

In this article, an approach for evolving software development processes using *Software Process Pattern* as a template of process evolving transformation is proposed. Software Process Pattern is a form of encapsulated knowledge about development project/product management issues. As well as Software Design Pattern, Software Process Pattern mainly consists of several components such as Problem, Context, Resulting Context, etc.

By applying pattern-based transformations to a primitive process, we can generate practical development processes with less effort of process authoring.

1. Introduction

In order to produce large-scale software with high quality in specified schedule, precise planning and appropriate management of projects are essential. This issue is widely discussed as software process technique. Various software process models or languages are proposed for process description, and some of them tried to describe practical process of real projects. These researches exposed a problem that huge efforts are required to clarify, define, and describe practical software processes, even though the modeling frameworks provide rich concept for description. Some process modeling framework offer capability to *reuse* existing process descriptions and to adopt/customize them, so that the process model fits the actual process requirement factors such as product scale, organization scale, organization culture, or applied methodology. However, only with these formal frameworks, it is not easy for process engineers to determine how customize target process model to fit the current situation.

Various knowledge and techniques of software process such as task scheduling, resource assignment, and product management, are actually required to build-up detailed process for practical projects. Skilled process engineers or project managers would have these techniques as their knowledge, and they can use this knowledge for customizing the target process. We believe some of process technique can be explicitly clarified and categorized, so that inexperienced process engineer can easily reuse them to customize their target process.

In this article, we try to formalize such knowledge as *process patterns*. We also try to facilitate these patterns for *process evolution*. In this approach, process patterns are used as templates of process evolving transformation. As an initial stage of this research, we have examined an existing process description, which was actually used for experimental software development project.

2. Software Process Patterns

2.1 Concept

Process pattern concept directly comes from *software design pattern,* which originally came from building-architectural patterns. There are already some researches concerning process patterns. Coplien has made some good works on organizational patterns in software development [1,2], and he called them "Process Patterns". As the other example, Ambler explicitly distinguishes organizational pattern from software process pattern in his book "Process Patterns"[3]. He defines the Organizational pattern as a pattern that describes a common management technique or a potential organization structure, while the Process pattern is defined as a pattern which describes a proven, successful approach and/or series of action for developing software. Ambler tries to define a software process for object-oriented development as a set of *process patterns* in his book. Thus these patterns are limited to object-oriented software development, although some of them seems to be valuable for the process using other methodology.

In this paper, we'd take more comprehensive view about *process patterns*; some patterns can be independent from certain methodologies, and they might be generic for various processes. We also argue that we take more descriptive approach about software process. We describe patterns based on diagram describing relationships among three entity types (task, role, and product). Detailed definition of our process patterns will be shown in following section.

2.2 Related Works

Other existing process patterns also identify valuable knowledge of software development. However, when we try to employ these patterns in order to construct practical software process description, there are many difficulties due to following characteristics:

- Many of them are organization management issues, not really process issues.
- There are so many ambiguities in pattern descriptions.
- There are no process formalisms or process models.

In next section, we introduce more explicit and formal process model, as a basis for clearer pattern definitions.

2.3 Process Pattern Language

We use following template for process pattern description:

- Problem (to be solved by using the pattern)
- Forces (additional factors for pattern needs, restrictions)
- Context (describing the situation for pattern capability) and Resulting Context (situation to be established) including:
  - Process diagram (E-R-A diagram)
  - Context Vector which represents project specific characteristics as a set of parameter values
- Description (actual pattern description in natural language)
- Remarks (other comments including rationale, classification hints, related patterns, etc.)

Table 1: Process patterns extracted from SFB501 reference process model description

ID	Name	Rough Categorization
SFBP1	Waterfall	Process phasing
SFBP2	Divide & Integrate	Generic strategy
SFBP3	Prototype	Requirement analysis method
SFBP4	Product Verification	Quality management principle
SFBP5	Feedback by Change Products	Process control principle
SFBP6	Requirement Analysis with Informal Object Design	Object-Oriented method specific task refinement
SFBP7	Two Phased Informal Object Design	Object-Oriented method specific task refinement

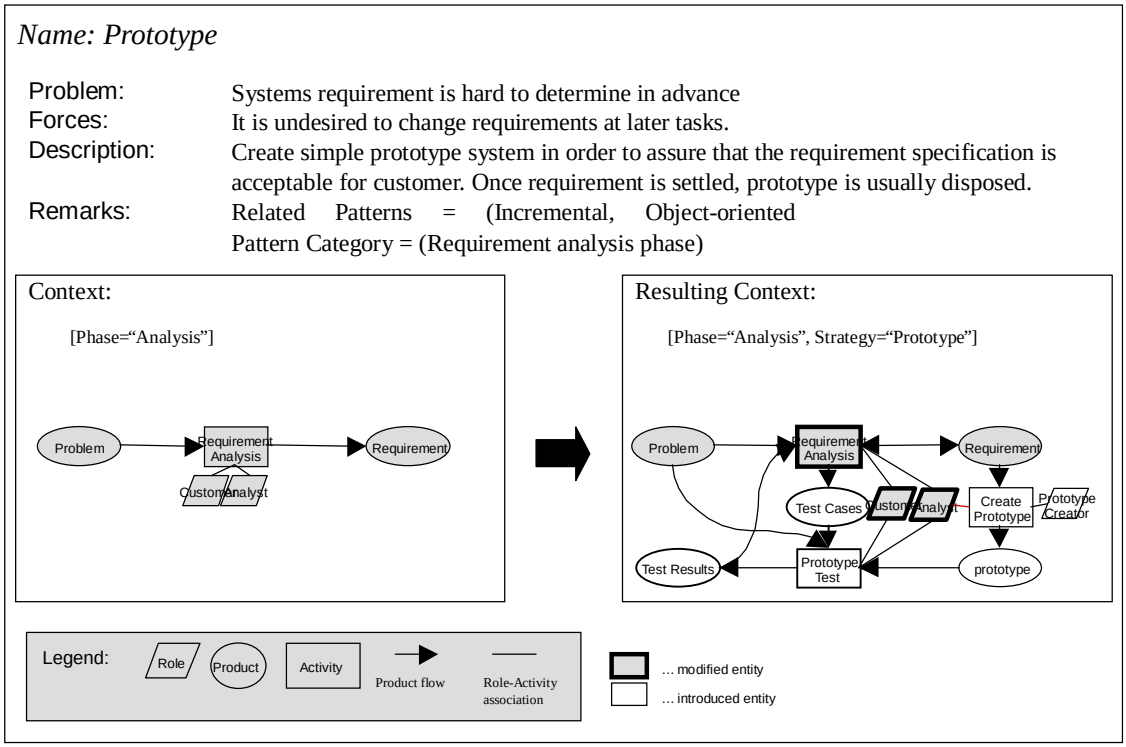

Figure 1. Example of Software Process Pattern Description (*Prototype* pattern)

An actual process patter example is shown in Figure 1.

2.4 Process Pattern Extraction

As a source of process patterns, we have investigated the description of SFB501 reference process model developed in University Kaiserslautern[4]. SFB501 is a special research project, and one of its goals is the development of methods and tools for experiment-based modeling of software engineering processes as well as the management of experimentally gained experiences.

An experiment to develop an air-conditioning control system based on prescriptive process has been carried out. The source software process description is written in a process modeling language MVP/L. The description contains 2300 lines including comments and 67 entities in total. *Gem* tool, a visual editor for MVP/L was mainly used for the investigation.

Table 1 summarizes extracted patterns from SFB501 process description. Figure 1 shows the content of *Prototype* pattern.

3. Process Evolution with Process Patterns

3.1 Approach

We now introduce the concept of software process evolution established by applying process patterns. We categorize process evolution in two cases --- process growth and process improvement. Process growth is made by applying process patterns to a certain primitive process in a generative way. Process improvement is done by finding *anti-patterns* that indicate some inappropriate structure and then applying improvement pattern to that process. For both cases, formal context matching should be done to determine the applicability of patterns. In this paper, we show an example of process growth.

```
procedure Main(top: process)
begin
  grow(top);
end

procedure grow(p: process)
begin
  /* decompose into sub-processes */
  determine_development_methodology(p);

  /* add management process elements */
  determine_management_methodology(p.sub[]);
  /* Go down and repeat */

  if (p.sub <> empty)
    foreach s in (p.sub)
      grow(s);
end
```

Figure 2. Meta-Process for Process Evolution

Figure 2 is an overview of meta-process for pattern-oriented process evolution written in pseudo language. In this case, methodology is first decided and the process will be decomposed into sub-processes, then management principle is decided and the process will be modified by adding some entities for management activity. Decomposition and modification are guided by process patterns. Earlier decisions, which were made at upper-level meta-process, will restrict later decision. This information is kept in context vector of the process and also in the *shape* of the process.

3.2 Evolution Example

Figure 3 shows step-by-step example of process growth established by pattern application. This example demonstrates how simple patterns can grow the process into more complex one.

4. Summary

We have proposed a framework of formal software process patterns, which is used to generative process evolution. We have actually extracted several process patterns from the existing process description, and we have also shown that these patterns are capable for generative process evolution. With this pattern framework, process patterns can be electronically

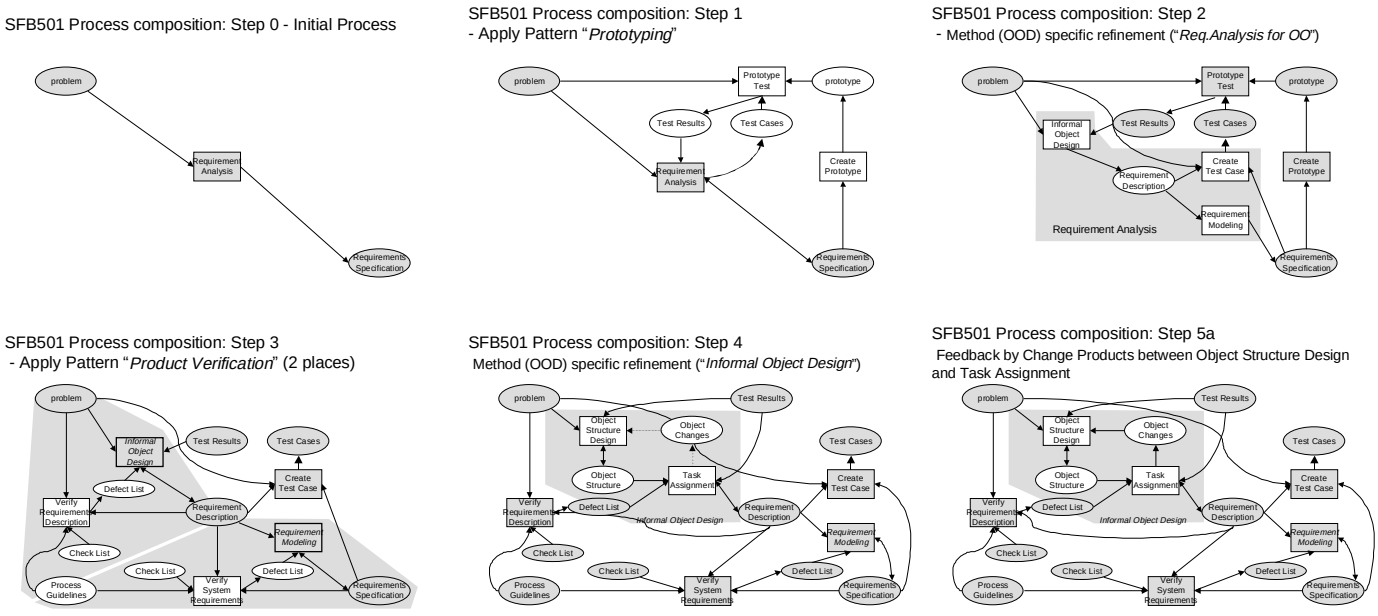

Figure 3. Example Process Evolution by Pattern Applications

stored and can be easily reused. In order to reuse stored patterns efficiently, appropriate categorization should be made. We are currently working on this categorizing issue. As a future plan, we are going to develop pattern-oriented process evolution system.

Acknowledgement

This research was initially done during the author was at University Kaiserslautern. The author is quite grateful to Professor Dieter Rombach, Dr. Martin Verlage, Mr. Juergen Muench and other people for their support and collaboration.

Bibliography

[1] Coplien and Schmidt (Ed.) "Pattern Languages of Program Design", Addison-Wesley 1995.

[2] Coplien, J.O. "A Generative Development-Process Pattern Language". Pattern Languages of Program Design, Addison Wesley Longman, Inc., pp. 183-237, 1995

[3] Scott W. Ambler "Software Process Patterns", Cambridge university press 1998.

[4] SFB501:Development of large systems with generic methods, WWW document, http://www.sfb501.uni-kl.de/