Chapter IV

Patterns in Software Maintenance: Learning from Experience

Perdita Stevens
University of Edinburgh, Scotland

In software design, patterns—that is, structured, named descriptions of good solutions to common problems in context—have become a popular way of recording and transferring problem-solving expertise. The aim of this chapter is to describe how patterns can help in the field of software maintenance.

There are two main uses of patterns in software maintenance. The first is to consider the introduction of a design pattern to an existing design. The second approach is to consider patterns for the maintenance process itself. For example, reengineering patterns describe solutions to common problems in the reengineering of a system, considering issues such as how to split the necessary changes into risk-minimizing steps. We discuss the advantages, pitfalls and practicalities of using patterns in both of these ways.

INTRODUCTION

Software maintenance is widely seen as the “poor relation” of the software engineering world. The skills that it requires are less widely recognized than those needed by more glamorous areas such as design or even testing. Few books on software maintenance are available, and those that exist are seldom seen on bookshops’ shelves. Software maintenance is gradually being recognized as an important area to cover in software engineering degree courses, but even now it is often covered in just a few hours of teaching. The overall effect is that maintainers of software generally acquire their skills almost entirely on the job.

This chapter appears in the book, Advances in Software Maintenance Management: Technologies and Solutions, edited by Macario Polo, Mario Piattini, and Francisco Ruiz. Copyright © 2003, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.
It is difficult to come up with a convincing explanation of why this should be so, and fortunately that is not the focus of this chapter. Rather, we focus here on how to move forward from where we are; taking the absence of a large, commonly-agreed upon body of software maintenance knowledge as given, how can an organization nevertheless ensure that it makes best use of the knowledge and experience available to it? In particular, we explore how people and organizations undertaking software maintenance can use patterns as a technique to help them learn from experience—either their own experience, or that of others.

What do we mean by a “pattern” in this context? We will go into more detail in the next section, but for now it suffices to say that a pattern is a named, structured description of a good way to solve a common problem. It’s a codified version of somebody’s experience, or preferably of several different people’s experience, with all the benefits and drawbacks which that implies, which we will discuss later. It typically takes up a few paragraphs to a few pages of text. An organization that is “using patterns” may be consulting patterns that have been written elsewhere, or it may be using the writing of patterns as a way of recording its own lessons learned, or (perhaps best) both. It is important to realize that in either case, the use of patterns will not constitute a complete methodology; you would not expect to welcome new recruits by giving them the company’s pattern library and nothing else. Patterns are most useful when they are used alongside a company’s existing methodology and process and integrated with it, and their use can be integrated with just about any process that a maintenance organization might use. Later in this chapter, we will return to the question of how to integrate the use of patterns with your maintenance process.

It is the flexibility and scalability of pattern use—the fact that a little or a lot of expertise may be recorded as patterns, and the fact that pattern use can supplement, rather than replace, what already works—that makes patterns interesting in the area of software maintenance. Whatever your organization’s software maintenance maturity, this chapter hopes to convince you that it is worthwhile to consider using patterns in taking the next step. Thus the objectives of this chapter are:

• to introduce the idea of patterns;
• to explore how this idea may apply in the context of software maintenance;
• to suggest some practical ways of proceeding, along with some pitfalls to avoid; and
• to provide leads into the wider literature on the subject for the reader who wants to explore further.

The remainder of this chapter is structured as follows. In the next section we will briefly introduce the ideas of patterns, including the process patterns that will be particularly relevant to software maintenance. We will also discuss the closely related topic of refactoring. Then we discuss how these general techniques apply
Related Content

Important Issues in Software Fault Prediction: A Road Map
www.igi-global.com/chapter/important-issues-in-software-fault-prediction/108634?camid=4v1a

The Future of Software Development
www.igi-global.com/chapter/future-software-development/29971?camid=4v1a

Model-Driven Configuration of Distributed Real-time and Embedded Systems
www.igi-global.com/chapter/model-driven-configuration-distributed-real/49156?camid=4v1a
Supporting Customizable Business Process Models Using Graph Transformation Rules
Verena Geist, Christa Illibauer, Christine Natschläger and Robert Hutter (2016).
International Journal of Information System Modeling and Design (pp. 51-71).
www.igi-global.com/article/supporting-customizable-business-process-models-using-graph-transformation-rules/170519?camid=4v1a