Patterns link People, Process, and Design

Position paper for the workshop “Beyond Design: Patterns (mis)used”, OOPLSA 2001

Klaus Marquardt
Käthe-Kollwitz-Weg 14, D-23558 Lübeck, Germany
Email: marquardt@acm.org

Writing patterns and submitting them to a conference is one of many possible ways to meet fascinating people – is that a misuse of patterns?

The other purpose I use patterns for is to explore and explain the relations and links between people, techniques, process, and management. One obvious situation occurred to me when a project had to define a communication protocol between different processors in a proprietary embedded system.

Project History

The team started to define the application level protocol on top of a field bus system according to known use cases. This was done well in advance of any implementation; the main purpose was to separate the code that run on different processors, divide the team into processor-specific sub-teams, and share only the message structure definition among the different teams.

The experiences were somewhat mixed. The sub-teams enjoyed being able to get their particular job done in separation, but no focus was put on early integration. Most projects experienced a phase late in the project when all subsystems had to cooperate, and the protocol turned out to be interpretable in different ways. Especially the responsibilities of the different processors were subject to lengthy arguments because they were not covered by the protocol structure definition.

To minimize the integration risk and to avoid these conflicts in the future, management decided to switch to object-oriented development. They had learned that OO would focus on responsibilities, and that OO could foster code reuse that in turn would smoothen the integration process.

Patterns Misused

In this project I became the system architect, and I used patterns in two different ways. First, I used them in a classic way, to explain object-oriented design principles. The design patterns were well accepted as most engineers recognized their own solutions in them, or understood how they could solve their current problems.

But process changed more significantly than "just" switching from C to C++. The order of engineering tasks became reversed in some respect, which irritated both developers and managers. The protocol definition was no longer based on public structures, but on serializable application objects, remote calls and replication. The application objects became shared code among different processors, and their evolution defined the contents of the replication based protocol parts. Remote calls to objects via Proxies were introduced to clarify the responsibilities of the different systems beyond mere replication.

The project participants recognized three aspects of this approach that strongly influenced their daily work. There no longer was a single owner of all the code executed on one processor. Also, the accustomed way of working was changed significantly, and they became insecure on how to proceed. Finally, it became less obvious when a task was completed, and what was needed to complete it.

Here comes the second use of patterns into place: to show all project participants the forces, benefits and liabilities of the seemingly pure technical solution. Actually this solution influenced all aspects of system development, including tracking, quality assurance, and maintenance. The patterns made the decision background clearly visible, and helped both developers and managers to see how to proceed and to make further, consistent decisions.