JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2006**

Vol. 5, No. 8, November–December 2006

Natures and Perspectives

John D. McGregor, Clemson University and Luminary Software LLC, U.S.A.

Abstract

"Nature" and "Perspective" have similar meaning for Eclipse developers and for personnel in a software product line organization. The nature of product line work differs from one part of the organization to another. A developer using Eclipse switches among perspectives to have ready access to the specific tools needed for the nature of the current work such as testing, modeling, or plug-in development. The product line developer has many reasons to switch among different perspectives on the product line. In this issue of Strategic Software Engineering I will explore the fundamental natures in a product line organization and the perspectives required for the success of a software development organization.

1 INTRODUCTION

I was working on a new plug-in for Eclipse the other day and was switching between the AADL (Architecture Analysis and Design Language) modeling perspective and the plug-in development (PDE) perspective when I was reminded of the switching between perspectives that occurs in a software product line organization. In the Eclipse IDE, a perspective groups the workbench functionality, including multiple plug-ins, that is used to complete a specific task such as editing a Java program or debugging a Java program. In a software development organization personnel often are assigned multiple tasks that require looking at the same information in different ways, in other words, adopt a different perspective.

A perspective provides a point of view from which a particular set of information is viewed. A developer may use several perspectives, one at a time, over the course of a task. The perspective usually has a default configuration that provides the developer with a workbench containing a set of windows and tools. Developers switch from one perspective to another when the task they are performing require sufficiently different approaches, often in the form of different tool sets.

I use the AADL modeling perspective that has three different types of editors and several analysis plug-ins to develop the basic structure for my plug-ins. Then I switch perspectives and use the PDE perspective to focus on the details of the plug-in behavior. The modeling perspective provides a high-level view across the product. The PDE perspective narrows the developer's focus while deepening its reach.

Cite this article as follows: John D. McGregor: "Natures and Perspectives", in *Journal of Object Technology*, vol. 5, no. 8, November-December 2006, pp. 7-13
http://www.jot.fm/issues/issue_2006_11/column1

A nature is a project attribute that is used to identify a “specific life cycle behavior” with the project [Eclipse 06]. This attribute associates the project with a particular type or style of development. A project may have multiple natures, all of which coexist at the same time but that characterize different attributes.

For example, the Java Development Tools (JDT) associate a Java nature with each new Java project that is started. This may associate certain tools, types of editors and other resources that are used by Java projects. There are certain other attributes that also acocmpany this nature. The notion of a classpath, for example, is specific to a Java development environment. Since a nature is a project attribute, natures are seldom changed once selected unless a major re-orientation occurs.

So we have two attributes related to how work is carried out in Eclipse, one of which is relatively persistent and one of which can be quite transitory. The nature of a project does not usually change while a developer may change perspectives several times in a relatively short time.

These two terms have implications for a software product line organization. If we think of these terms as they apply to projects and their personnel, these relationships are about right. People, companies, and working groups seldom change their fundamental modes of operation but as we move from one task to another this often results in a shift in perspective, however temporary it may be. I am going to examine both of these ideas below.

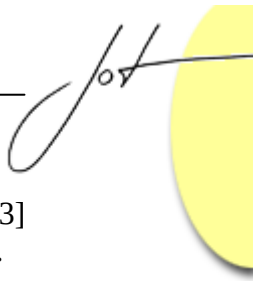
2 THE NATURE OF THINGS

People differ in their basic attitudes and outlooks on life. We often say a certain person is just naturally cheerful or outgoing. Software development projects, and their sponsoring organizations, have fundamental natures as well. The process model, e.g. agile, iterative, or waterfall, adopted by a project often reflects that organization’s nature or the nature of the application domain. A software product line organization also has a specific nature.

Core asset building

The nature of core asset building in a software product line organization spans the extremes reflected in the pure proactive and reactive approaches to asset base development with a contiuum of intermediate flavors available. The proactive approach builds a complete, or nearly complete, set of assets prior to the majority of product development. The reactive approach builds assets, or modifies existing assets, just in time for use in a product.

The proactive approach requires a project to have a nature that is reflective and encompassing and a rhythm that is sustainable for a relatively long period. Core asset builders must reflect on the overall goals and consider the complete spectrum of products in the product line. The asset development life cycle is usually highly iterative and incremental. It often takes several attempts to achieve the correct amount of flexibility



and scope. A process following the Rational Unified Process model [Kruchten 03] provides a good model for the core asset team to apply to each asset development effort.

The reactive approach to core asset building requires a nature that encourages extensibility and modifiability in the architecture and a development process with a rhythm that is relatively fast. Assets will change often, relative to the rhythm of the organization, and may need to be extended in unexpected ways. This nature relates well to the various flavors of agile process models [Paulk 02].

Product building

The nature of product building in a product line is a straightforward, no nonsense approach intended to get a product deployed as rapidly as possible. Product building is well-planned and efficient. This may be one of the few places where I have seen that a waterfall process model [Royce 70] might be appropriate. In fact, in many product lines, the product building process is so highly automated that the process has very few hands-on activities beyond product identification.

The nature of a product building project is influenced by the nature of core asset development. In an organization that has a proactive nature, product building is highly optimized; developers follow the prescribed production plan to reach a successful conclusion as rapidly as possible. A reactive nature of core asset development results in less optimized product development. The trade-off is based on the assumption that by waiting the new assets and the modifications to existing assets are more accurate and there is less wasted effort resulting from changes to existing assets.

The Nature of Nature

Because it is a relatively long term investment, people's attitudes toward their organization's nature are often very deeply rooted. A few years ago I participated in a panel on testing that included agile and non-agile supporters. This proved to be a much more spirited discussion than most panels because it addressed the basic nature of testing for those of us on the panel. Software professionals seldom view these basic natures as just tools in a toolkit to be used when appropriate. Nature often embodies some deeply held development philosophy.

Certainly the discussions about proactive versus reactive in the product line community are spirited. One possible compromise between the two extreme natures of proactive and reactive is to adopt a perspective on certain planning assets that is different from that for less encompassing assets such as individual product components, product test plans, etc. Being more proactive on the planning activities and product line wide assets, such as the software architecture, provides a solid foundation upon which program components, tests and other assets can be build more reactively.

3 PERSPECTIVES IN PRODUCT LINE DEVELOPMENT

One development perspective about which I have written before is the "testing perspective"[McGregor 02]. This refers to the view of a software product that a developer needs when developing and applying test cases. To me every role to which a professional is assigned requires a specific "perspective" on the development effort. That perspective encompasses all of the relevant information for that role. The perspective focuses the professional on what is important for their work.

The testing perspective requires objectivity, thoroughness, skepticism, and the ability to be systematic. The testing perspective focuses the developer on using specific criteria for selecting test cases, measuring coverage of the code under test to ensure thoroughness, and verifying rather than assuming the code possesses certain properties. The typical developer must switch between a "development perspective' and a "unit testing perspective." This switch will often occur many times in a single day. With test-driven development it occurs even more frequently.

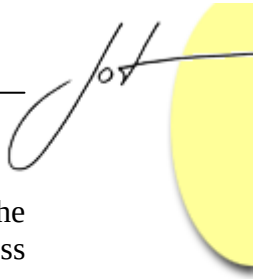
Switching between perspectives in Eclipse is a purely automated process that happens completely, consistently, and hopefully correctly, every time while the human perspective switch may not always be complete or consistent. It may be impossible for a human to totally change focus, particularly if, for example, they are testing their own code. The result of a partial switch is a developer who is less effective at their task. Part of the switch in perspectives is a switch of mental focus from building up to dissecting. The testing perspective requires a change from "how can I make this work" to "how can I make this fail."

The two principle activities in a software product line organization are core asset development and product building. These two activities require different perspectives on the core assets: designing to be reusable and designing by reusing.

Designing to be reusable

Designing to be reusable is a broad perspective with many possibilities. The product line core asset perspective is more narrowly focused than just a general "lets make this reusable" approach but much broader than the individual product design perspective. The product line design perspective includes something not in the typical designing for reuse approach: constraints.

The designing to be reusable perspective is constrained by the well-defined scope of the product line. The product line scope, which describes what products are in the product line and which are not, provides the product line reusable design perspective with a specific view of the limits to usage of the assets. The perspective includes tools such as feature trees and architectures with which the required variability is analyzed, traded off, and then realized [Czarnecki 00].



The designing to be reusable perspective is also constrained by the goals of the product line organization. Design decisions are part technical decisions and part business decisions. The perspective includes economic modeling tools that support decision making [Clements 05].

The designing to be reusable perspective is constrained by the production strategy of the product line [Chastek 02]. The designer of an individual core asset is not free to use any implementation technique they wish or to decide the scope of variability in their asset without regard to the larger issues of the product line. Ultimately the core asset must be compatible with other assets and must be usable in the product production process. This perspective includes method engineering tools that support the translation of the production strategy into core asset development environments.

The core asset designer must be able to view their work from the reusing perspective occassionally in order to understand the target audience's needs. This is often accomplished by developing prototype products that are built by reusing core assets such as the product line requirements model, software architecture, and components.

Designing by reusing

Designing a product by reusing narrows the developer's perspective to those options that are currently in inventory. It uses a search activity that is limited by the number of possible assets that fit a given situation and the number of optional implementstions at the variation points. The design is limited to available materials and so the amount of creativity is also limited. But, this is a good thing! Products need to be assembled and deployed rapidly.

The reusing developer must be able to view their situation from the perspective of the reusable designer to understand the intent in certain design details. No specification or documentation is sufficiently complete to capture the complete design rationale. Being familiar with the reusable perspective provides the reusing developer with insight into the intent behind the reusable design.

Facilitating Differences in Perspective

These two perspectives co-exist in a software product line organization. In addition to the need for personnel assigned to one role to understand the other roles in the organization, there is often a need to move personnel from one role to another.

1. You can negate the effect of changing perspectives by never changing perspectives. Have a separate team for each different perspective. Many companies have separate test teams even down to the unit level. Many software product line organizations have separate core asset development team and product development teams.
2. Provide sufficient process and tool support to guide the personnel through each activity, and as a result, each change in perspective. In a software product line core assets come with an attached process that provides just this type of support.

The attached process can focus a developer on the appropriate issues and establish the proper horizion for their efforts. Some product line organizations enforce constraints such as not making changes to core assets by delivering code-based assets in binary.

3. Rhythm can also help achieve the correct perspective. Grady Booch has spoken of how a regular rhythm helps keep a project on track [Booch 95]. I have used this approach for many years in my courses as well as consulting engagements. I assign homework on the same day of the week each week and each assignment has the same duration. Time boxing when developers switch perspectives provides a rhythm that will reinforce the change and help personnel focus more quickly on the current environment.

4 SUMMARY

I have explored the nature of product line activities and some of the perspectives the personnel must have on those activities. There are many more examples of natures and perspectives in a software product line. The Software Engineering Institute has identified three essential activities including the core asset development and product building that I have discused. They add “management” as an essential activity [SEI 06]. This requires another perspective.

I have also provided a few suggestions on ensuring that the organization has the appropriate nature and perspective at each moment. Usually this is an implicit event. It just happens. However, this is a strategically significant that should not be left to chance.

What makes this strategic? The product line strategy uses specific activities to achieve specific goals. The successful execution of a product line strategy requires that personnel focus on their responsibilities. To do this they must have the correct perspective on their assigned tasks and the processes that guide their work must impart the correct nature on the organization. Adopting the correct perspectives and achieving the appropriate natures is possible with planning, training, and tool support.

5 ACKNOWLEDGEMENTS

Thanks to John Hunt of Covenant College for insightful comments that greatly improved the paper.

REFERENCES

[Booch 95] Grady Booch. Object Solutions: Managing the Object-Oriented Project, Addison-Wesley, 1995.

[Chastek 02] Gary Chastek and John D. McGregor. Guidelines for Developing a Product LineProduction Plan, CMU/SEI-2002-TR-006.

[Clements 05] Paul Clements, John D. McGregor, and Sholom G. Cohen. The Structured Intuitive Model of Product Line Economics (SIMPLE), CMU/SEI-2005-TR-003.

[Czarnecki 00] Krzysztof Czarnecki and Ulrich W. Eisenecker. Generative Programming, Addison-Wesley, 2000.

[Eclipse 06] http://www.eclipse.org

[Kruchten 03] Phillipe Kruchten. The Rational Unified Process: An Introduction, Addison-Wesley, 2003.

[McGregor 02] John D. McGregor. A Practical Guide to Testing Object-Oriented Software, Addison-Wesley, 2002.

[Paulk 02] Mark Paulk. "Agile Methodologies and Process Discipline," Crosstalk, October 2002.

[Royce 70] Winston Royce. "Managing The Development of Large Software Systems," Proceedings of IEEE WESCON, v 26, n August, p. 1-9.

[SEI 06] Software Engineering Institute, "Framework for Product Line Practice," http://www.sei.cmu.edu/productlines, 2006.

About the author

Dr. John D. McGregor is an associate professor of computer science at Clemson University and a partner in Luminary Software, a software engineering consulting firm. His research interests include software product lines and component-base software engineering. His latest book is *A Practical Guide to Testing Object-Oriented Software* (Addison-Wesley 2001). Contact him at johnmc@lumsoft.com.