JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2010**

Vol. 9, No. 1, January-February 2010

Intentionality

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

Abstract

In philosophical circles intentionality has a very abstract meaning that relates to consicousness of the mind. I have a somewhat simpler view that "being intentional" means consciously deciding what you wish to do and then doing it. This includes following "best practices," but only those that apply to the current context. In this issue of Strategic Software Engineering I will discuss examples of how intentionality has been, and can be, used to improve products and organizations.

1 INTRODUCTION

"If you keep doing what you have always done, you will keep getting what you have always gotten," according to Special Agent Anthony Denozo of CBS television's NCIS. He said this as he "intentionally" chose to take a different path because he was tired of his life as it was. All too often we simply do what we have always done without first considering alternatives even though forces in the world may have made a once very good decision into what is now a very bad one. With the latest statistics claiming that 15 - 50% of projects are cancelled before they deliver anything, perhaps what we have always gotten is not what we have always wanted. In this issue of Strategic Software Engineering I will discuss being intentional in our work and careers.

Intentionality is a property of a decision. A decision that is the result of explicit examination of relevant factors is an intentional decision. It takes longer to make an intentional decision so many managers don't make the effort. They go with the default, business as usual, and they get what they have always gotten.

The same is true for defining development methods. We need to be intentional about how we choose to develop. Gary Chastek and I have developed a sequence of steps that a software product line organization can follow to determine the plan for producing products [Chastek 08]. The organization first determines a production strategy based on the strategic goals of the product line. The product line organization filters the strategic goals to identify those that relate to how the products are developed. The strategic production goals are associated with specific production techniques forming the production method. This is where it gets intentional. Rather than just doing what we have always done, we intentionally select only those production techniques that explicitly relate to the current goals of the product line. Finally the production plan is formed by creating a template schedule for carrying out the production method.

Like many researchers my age (old enough) I have seen many research areas blossom and I have seen many of them fade. Some faded because the most important problems were solved (basic issues of object-orientation) while others (AI) have faded because insufficient progress was made. A researcher has to be particularly intentional since we know that no problem will last a 30 year career. We have to make decisions to knowingly shift directions no matter how subtle the shift may be as new research areas emerge and old ones wane.

One manager in Microsoft made an intentional decision [Taft 09]. He determined that his organization needed to be more intentional about their engineering. He shut down development for several months and directed a bug hunt. They found the bugs by increasing the amount of automated testing they had available to use. Once the majority of the bugs in their current product were identified and repaired he put in place a new method. Visual Studio was the product. At one point during the development of Visual Studio 2005 32,000 defects had accumulated. By being intentional about maintaining the appropriate level of test automation the building of Visual Studio 2008 never had more than 5,000 defects identified but not found.

Bureaucratic intentionality is not always a useful thing. All of us have seen “This page intentionally left blank” in some official document. In this column I will avoid philosophical arguments about whether the page is actually blank once that notice is added or not; however, every time I see that phrase I am reminded of people following procedure whether it makes sense or not. I will try to remain focused on productive uses of intentionality.

In the next section I will discuss best practices in the context of intentional decision making. Then I will consider three critical areas: requirements, software architecture, and reuse. I will conclude with some observations cutting across these areas.

2 BEST PRACTICES

Many organizations claim to follow best practices and a few even publish what they term best practices. Perhaps I am just splitting hairs but “best” is an absolute that requires justification. Most of the published best practices have not been critically compared to any other practices much less all other similar practices. At best, “best” means best in the limited scope of that organization or that of the author.

Companies publish best practices for internal use and for use by their customers. For example, IBM has their famous Red Book series, but most of those are how to solve specific problems using IBM built tools. At best, these are well-thoughtout processes, but with no claim to the title “best practice.”

Just as I was writing this column a group of well-known software engineering researchers made public an effort to be more intentional about identifying best practices by establishing a theory regarding the foundations of software engineering. If this effort bears fruit, software engineers will be able to make more precise decisions about how to engineer processes.

Their effort, Software Engineering Method and Theory (SEMAT) is described at www.semat.org. They present obstacles to and objectives for their work. One point that I am certain they recognize, but have not explicitly stated, is that no technique is best, or even necessarily good, in every possible situation. To establish best practices, there needs to also be research on specifying contexts. Then practices can be evaluated in specific, and appropriate, contexts and compared fairly.

The context includes the culture and domain of the organization which the development occurs and of the users of the product. The rhythm at which the organization is operating also influences which practices are best. Intentional selection of practices will be far better than relying on “best practices” until we arrive at a method for actually determining “best”.

3 BEING INTENTIONAL ABOUT REQUIREMENTS

Over time I have seen many managers be passive about requirements: “Yes, requirements change late in the project. It has always been that way.” “Customers never know what they really want until they see what they don’t want” is heard often. With no claim to be defining a best practice, lets consider some techniques for requirements modeling.

There are several ways to represent requirements.

- For many years, and still in some domains, requirements were individual statements of required behaviors and qualities. The favorite tool was a spreadsheet. This form is very modular. It is easy to change one requirement but it is virtually impossible to determine the impact of that change on other requirements.
- Then the use case technique introduced actors and scenarios [Jacobson 92]. By intentionally identifying the set of actors, a use case model could be evaluated for completeness, which had eluded previous requirements techniques. The addition of scenarios gave a basis for structuring and classifying requirements. This in turn provided a basis for determining the impact of a change and determining whether the requirements were consistent.
- Agile development methods introduced stories as a means of capturing requirements. [Cohan 04] suggests a format for user stories:

  As a (role) I want (something) so that (benefit).

  These stories are less formal than the scenarios in use cases and less structured. Each story is assigned a priority which provides a sequence but does not support impact analysis.

Most recently software product line people have used feature models as at least part of the representation of requirements [Kang 90]. Feature models add the concept of variability so that the features can represent the requirements of several products, not just one. Combined with a constraint language to identify dependencies among features, this model supports a number of analyses.

I now use a combination of feature modeling for high level product specification and use cases for more detailed modeling of requirements. I intentionally combined these techniques because neither provides everything needed for the total development cycle but together they do.

The actor model of the use case model is easier to evaluate for completeness than a list of requirements and the feature model is easier to evaluate for consistency. I proactively conduct this analysis and then actively work to complete the use case model.

4 BEING INTENTIONAL ABOUT ARCHITECTURE

Grady Booch has said, “An intentional architecture is explicitly identified and then implemented. [Booch 06]” Software architecture has gone from being a “black art” to an engineering discipline because of the intentional techniques developed by the software architecture research community. Previously I described some of the techniques in the modern theory of software architecture so I will not go into much detail here [McGregor 04]. What I will do is list several items that illustrate how intentional architecture practice has become.

Quality attributes – The recognition of non-functional requirements is no recent discovery, but earlier methods left much of the responsibility for satisfying these requirements to developers rather than architects. Developers often coded a system and then evaluated its performance. If it was not acceptable they then attempted repairs. Recent techniques have become much more intentional. Attributes such as performance are now intentionally included in the architecture design and models are used to predict the level of that quality in the final products. The Architecture Analysis and Design Language (AADL) provides an extensible set of properties that are used to represent specific quality attributes [AADL 09].

Architecture tactics – An architecture tactic is a design decision that affects the value of a quality attribute [Scott 09]. The software architecture initiative of the Software Engineering Institute has defined a standard approach to representing a tactic and has identified a number of tactics. Arche is an experimental tool that allows the user to explore their architecture in terms of two specific quality attributes: performance and modifiability [Bachmann 03]. The tool allows users to add other quality attributes by defining their own plug-ins to Arche. Each reasoning framework associates a means of predicting the level of the quality attribute with specific tactics that enhance the level of that attribute.

Architecture documentation – Techniques for identifying multiple views and standards for the content of those views have made architecture documentation more effective and more complete. Explicit directions for documenting, such as Views and Beyond [Clements 02] and the IEEE 1471 standard [IEEE 09], allow the architect to explicitly address the needs of individual stakeholders.

Architecture evaluation – The Architecture Trade-off Analysis Method (ATAM) is used to evaluate an architecture [Barbacci 03]. The evaluation technique identifies scenarios that address each of the quality attributes specified in the requirements. The scenarios in the use cases are a starting point for the development of these scenarios. The evaluation can be focused in any way that supports the goals of the organization.

Beyond these ideas, there is a convergence that will make architecture practice more intentional. Architecture description languages are maturing to the point that the architect can make models to aid their thinking. Those models of the architecture can be used to analyze and predict quality attribute levels. That same description can be used as the basis for the documentation and evaluation.

5 BEING INTENTIONAL ABOUT REUSE

Over the years many schemes have been defined for reusing software from one product to another. Most of these schemes have failed. Those failures are the result of unintentional choices that accept the status quo rather than intentionally exploring techniques that address explicit objectives. For example, everyone has agreed that reuse works best within a specified context, but most projects are single development projects. There often is not sufficient continuity or prior planning to retain a sufficient amount of context from one single-product project to the next. A software product line organization has several characteristics that help the organization make intentional decisions about reuse and operate as a single context:

An explicit scope – A software product line organization explicitly chooses the limits on the set of products it will construct. This scope is represented by a feature model. The scope provides the context needed to make a software reuse approach successful.

Architecture-centric – The architecture drives many of the decisions in a software product line organization. In particular, the architecture provides a context in which choices can be made explicitly.

Explict production plan – As described in the Introduction, a software product line organization intentionally analyzes its goals and derives a production method. The process of developing the production strategy, production method, and production plan stimulates an intentional approach.

Hewlett Packard was intentional about the design of their reuse organization, but in an intriguing way. They intentionally put in place an approach they termed “Progressive Open Source [Melian 02].” However, to be successful the approach had to be allowed to grow organically rather than following a mandated approach. They intentionally started a process that is not intentional! This is consistent with management reducing their level of control of an open source development process.

6 SUMMARY

Being intentional is necessary for being strategic. Strategic planning requires intentional actions to reach specific goals. Many software engineering methods have taken what used to be intuitive techniques and made them more intentional. Strategic software engineering is inherently intentional and will get you where you plan to go. Just be certain that where you plan to go is where you want to go.

REFERENCES

[AADL 09] www.aadl.info, 2009. Last visited 12/5/09.

[Bachmann 03] Felix Bachmann, Len Bass, and Mark H. Klein. Preliminary Design of ArchE: A Software Architecture Design Assistant, CMU/SEI-2003-TR-021, 2003.

[Barbacci 03] Mario R. Barbacci, Paul C. Clements, Anthony J. Lattanze, Linda M. Northrop, and William G. Wood. Using the Architecture Tradeoff Analysis Method (ATAM) to Evaluate the Software Architecture for a Product Line of Avionics Systems: A Case Study, CMU/SEI-2003-TN-012, 2003.

[Bell 07] Peter Bell. A practical high volume software product line, Conference on Object Oriented Programming Systems Languages and Applications, 2007.

[Booch 06] Grady Booch. The Accidental Architecture, vol 23, n 3 IEEE Software, 2006.

[Chastek 08] Gary J. Chastek, John D. McGregor: Production Planning in a Software Product Line Organization. SPLC 2008: 369.

[Chesbrough 03] Henry Chesbrough, *Open Innovation: The New Imperative For Creating and Profiting from Technology*, Boston: Harvard Business School Press, 2003.

[Cohan 04] Mike Cohan. User Stories Applied: For Agile Software Development, Addison-Wesley, 2004.

[Clements 02] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, Judith Stafford. Documenting Software Architectures: Views and Beyond, Addison-Wesley, 2002.

[IEEE 09] IEEE 1471, Recommended Practice for Architectural Description of Software-Intensive Systems, http://www.iso-architecture.org/ieee-1471/, last visited 11/29/2009.

[Jacobson 92] Ivar Jacobson. Object Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, 1992.

[Kang 90] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study, CMU/SEI-90-TR-021, 1990.

[McGregor 04] John D. McGregor: “Software Architecture”, in *Journal of Object Technology*, vol. 3, no. 5, May-June 2004. http://www.jot.fm/issues/issue_2004_05/column7/

[Melian 02] Catharina Melian1, Cathy Burles Ammirati 2, Pankaj Garg, Guje Sevon. **Building Networks of Software Communities in a Large Corporation.** www.hpl.hp.com/techreports/2002/HPL-2002-12.pdf, *last visited 11/29/2009.*

[Scott 09] James Scott and Rick Kazman. Realizing and Refining Architectural Tactics: Availability, CMU/SEI-2009-TR-006, 2009.

[Taft 09] Darryl K. Taft. Microsoft pays the Visual Studio Debt, DevSource, http://www.devsource.com/c/a/Using-VS/Microsoft-Pays-Visual-Studio-Debt/, last visited 11/29/2009.

About the author

Dr. John D. McGregor is an associate professor of computer science at Clemson University, a visiting scientist at the Software Engineering Institute, 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.