JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2007**

Vol. 6, No. 8, September-October 2007

Form over Substance

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

Abstract

Beware of the colleague or supplier who spends large amounts of time in meetings discussing the format, sequence, and wording of documents they will deliver and very little time on the actual content. Strategically, substance is what counts. In this issue of Strategic Software Engineering I will point to some common problems when form becomes a higher priority than substance and what value is added when these problems are addressed.

1 INTRODUCTION

The motivation for this column came as I sat on a teleconference and listened as a vendor read the outline for a document that we all had copies of. The vendor's customer asked several pointed questions about the content. The vendor tried mightily to avoid the questions by proceeding on with the recitation of the form of the document. The vendor was late and had no content but was trying to divert attention away from that deficiency by focusing on the form of the document.

Clients who are using a model-driven approach for the first time always want to know what a good model should "look like." They wonder how they will know when they have drawn "enough" diagrams. They want to know how many objects should be in the model. They feel that by knowing the desired form they can fill in the blanks and avoid troublesome issues. I don't blame them. I often start a paper by putting in the essential elements for whatever format the conference or journal wants, hoping that the content will follow.

This is like the student, who when assigned a paper to write, immediately asks, "How many pages should it be?" My standard answer, delivered through clenched teeth, is, "As long as it takes to tell the story, but no longer."

One of my colleagues in Luminary Software became very angry when she realized that the project, which she was mentoring, intended to pass their ISO 9000 audit by have a "forms party." In one night they completed all the paper work for several months worth of development in an assembly line of filling out and signing inspection reports and other

Cite this column as follows: John McGregor "Form over Substance", in *Journal of Object* Technology, vol. 6, no. 8, September - October 2007, pp. 9-17 http://www.jot.fm/issues/issue_2007_09/column2

documentation. Not only could they not understand why she was upset, they couldn't understand why the project was in trouble.

Lets be clear. I am not saying that the format of written processes shouldn't be carefully considered or that companies shouldn't spend time creating templates. I felt very good on a recent Sunday riding home from Ireland on a Boeing 767 knowing that the pilots have standard forms that must be filled out, checklists that must be completed, and processes to follow. But I would not feel so comfortable if I thought the pilots waited until after takeoff to fill out the forms and then simply put an answer in every blank without being concerned about what the actual data was.

Substance is hard. You have to think. You can't phone it in. Form you can copy from a book or a website. Substance requires indepth analysis and design. It requires consideration of a wide range of issues and managing a number of tradeoffs. Form is tactical, substance is strategic.

Substance is often invisible and difficult to measure. Form is visible. Racks of procedural manuals and the minutes of review meetings are tangible. They can be counted. The fact that no one ever reads, much less follows, the wonderful processes or that nothing was discussed at the review meeting except how to complete the inspection report, is not as obvious and seldom seen as the cause for project difficulties.

2 FORM FOLLOWS FUNCTION

Form follows function is a ubiquitous design rule which would seem to support substance over form. It simply sets the stage for development processes that first determine what a product should do and then determines the shape of the architecture. And yet how many projects start with an assumption about which architectural style will be used? That a service oriented architecture is the right form for their project? I refer to this as "magazine cover" architecture. That is, the architecture is picked by what is most popular at the moment, not what is the best structure for the product. I vividly remember the client who had designed a transaction oriented architecture for a product that spent most of its time serving up streaming video. That turned out to be a strategic error delaying delivery and reducing market share.

In a strategic design process the form of the product is derived systematically from the functional and non-functional requirements – the function. I have already written about the strategic value of software architecture [McGregor 04] so I won't repeat myself. Others have written about how to derive the architecture [Bass 03] and I will not repeat them either.

The derivation of the architecture from the requirements is hard work. But, there is a reward. The act of creating the architecture is probably the best requirements verification tool there is. The feedback from the architecture definition ensures more thought will be put into the requirements than would otherwise be the case.

3 SEPARATE BEHAVIOR FROM STRUCTURE

Form follows function can have unexpected consequences as a group of products evolves. As the functions of a system evolve, there will be pressure on the architecture to follow, but changes to the architecture can send ripples through a project. If the architecture has been properly abstracted that pressure will be minimized. The software product line environment actually facilitates this approach. By building an architecture that spans the range of systems in the product line, the architecture will be more flexible. This is one case in which form shouldn't follow function any more than necessary.

Architecture Definition is an excellent time to be certain that the structure of the product is sufficiently robust, within the scope of the set of products, so that anticipated changes in behavior will not require structural changes in the product. I liken this to the need to minimize the number of load bearing walls in a building. Structures that span a space with no supports inbetween are much more flexible as to the functions that can be carried out in the building.

4 STRATEGICALLY SIGNIFICANT SUBSTANCE

My company, Luminary Software, adopted an approach to describing processes in which two types of input and two types of output are listed for each phase, see Table1 for an example. We termed these "actual" and "by-product." [Russ 00] For the domain analysis phase, shown in Table 1, there is an actual input of " an understanding of what the system should be ". The by-product input is a business plan, which follows a standard form. By taking this approach we tried to draw attention to the intangible – an understanding - which is often ignored but is actually the substance for which development personnel should strive.

This illustrates one of my concerns about many processes, some of which are used in companies with high CMMI classifications. A good process is one that describes exactly what an expert would say is necessary to accomplish the task. A good process is one that is so natural you only have to read it once and then you can repeat it many times. It has a form that fits the substance in a comfortable way.

Good organizations recognize the value of substance. The Software Architecture Technology (SAT) initiative at the SEI has made extraordinary progress in defining techniques that incorporate the non-functional qualities of a software architecture into the architecture design process [Bass 03]. These qualities have been thrown around in software development for many years. Everyone claims their systems are maintainable, but no one could prove it. These architecture definition processes provide a natural way to talk about a substantial portion of a product's requirements.

5 PLANNING VS PLANS

Dwight Eisenhower has been attributed with saying that he thought plans were useless but that planning was essential. All too often managers choose to develop plans by assigning one person to “write” the plan, but this usually means create the plan as well as write it. Often this person does not have any stake in how well the plan works and may have never carried out this type of work.

1. DOMAIN ANALYSIS

1.1. Description

To capture the concepts and relationships within the bodies of knowledge that underlie the basic problem to be solved by the application. Operating within the scope defined by requirements, domain analysis provides a superset of concepts and relationships needed for application analysis.

1.2. Responsibility

- *CONCEPTUALIZER* assists in determining the domain boundaries and serves as a domain expert.
- *DOMAIN EXPERT* provides concepts and connections between the concepts.
- *MANAGER* reviews the output of this phase to be able to produce schedules, resource allocations, and the software development plan.
- *SYSTEM ENGINEER* provides validation during the assessment activity.
- *ARCHITECT* gains an understanding of the domain to construct the appropriate architecture.
- *DEVELOPER* takes the lead in soliciting domain expert input and defining and modeling the domain.
- *TESTER* guides the assessment of the domain analysis by-products ensuring that they are testable and non-contradictory.

1.3. Input

1.3.1. Actual

- An understanding of what the system should be
- The collective domain knowledge of those participating in the analysis

1.3.2. By-product

- Business plan describing the concept or idea and the requirements

1.4. Entry Criteria

Agreement has been reached that the requirements are sufficiently scoped to begin this phase.

1.5. Activities

- Domain Modeling using UML models.
- Guided inspection of the domain model.

1.6. Output

1.6.1. Actual

- An understanding of specific domains related to the problem being solved.

1.6.2. By-product

- A UML model capturing the concepts and relationships
- Mapping between the domain model concepts and the use case model

1.7. Exit Criteria

1.7.1. Proceed to next phase

- If the results of the guided inspection show the domain model is complete, correct, and consistent
- If the requirements model is still sufficiently complete

1.7.2. Iterate back to previous phases

- REQUIREMENTS: If domain concepts have been discovered that lead to the discovery of new uses of the system

1.8. Metrics

1.8.1.	**Raw data** ➢ Number of concepts (classes) captured in model
1.8.2.	**Derived measures** ➢ Percentage of use cases covered by concepts in the domain model

Table 1 - Domain Analysis Phase Description

The substance in planning is the thought that goes into alternatives and constraints. Which specific tasks will best solve a specific problem and what constraints make them the best choice? A plan seldom captures this level of thinking nor does a person reading a plan think through as thoroughly the issues as happens during planning. Having the people who will do the work create the plan ensures that they have thought through some of the important issues. Then, the form in which the plan is written or even if it is written no longer matters.

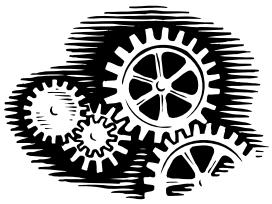
Open source development projects and many agile development efforts do a remarkable job of planning by staying in touch with substance without being sidetracked by form. Both of these strategies emphasize planning over plans. Teams using the Scrum agile method plan everyday but seldom produce written plans. This works since they never plan a very large piece, content or time wise. Planning occurs in 24 hour increments.

One reason there is more substance than form is that there seems to be a very shallow management hierarchy on most open source projects and those positions are filled based on merit – managers beware. That is, leadership positions (different from management positions) are filled by persons who have in the past made the greatest contributions. Therefore, they know the demands on those they lead – they've been there. They also would like to continue to contribute so they do not allow overhead to build up that they will have to manage.

Agile projects also keep a small management layer. The developers are self-organizing. They keep focused on substance by very frequent, in some cases daily, interaction with their customer. It is certainly easier to focus on making real progress when the customer is that close. I have seen projects where the interaction with the customer is either indirect at best or only about once a year. There is a very different atmosphere in that project office from ones where the customer has a presence in the same building.

The Product Line Systems program at the SEI defines 29 practice areas that a product line organization must address.[SEI 07] Some organizations assign individuals to each of the practice areas and have them write a process for each practice area. The product line manager has a checklist of practice areas and most reviews, status reports, and other oversight is based on a practice area by practice area approach. They miss the point. The strategic substance is in the interactions among the practices and how those interactions represent a substantial amount of intellectual property.

6 WHEELS WITHIN WHEELS PROCESS MODEL



Most projects are sufficiently complex that the top level process is really a matter of choreographing the interplay among numerous detailed processes. The "wheels within wheels" process model allows me to blend the strategic and tactical perspectives and other perspectives inbetween. By wheels within wheels I mean there is an overarching process that encompasses other, more narrowly focused, processes. The processes are iterative, hence the wheel analogy.

A large project model such as the Rational Unified Process model can form the basic process model. [RUP 07] The agile approaches to development have much to commend them, but I am reluctant to use them on very large, 100 plus headcount, projects. Large projects such as these need a structure that has been thought through before modules are constructed to flesh it out. The wheels within wheels gives the correct perspective in which an overarching process can manage other more flexible processes.

In a product line organization, there is an overall production process that coordinates the development of core assets and the development of products. Then individual product teams may use an agile product development process within the context of the overall process. Another product team could perhaps use a somewhat different process to realize their product. Production planning gives the organization the opportunity to think through these issues.

Within each development phase there is a process that operates each activity. Even though there is a requirements phase and process for the product line, there is also a product level requirements process that starts with the core asset requirements model and produces the model for the individual product. One of the critical success factors in a software product line organization is the ability to coordinate a single process across the core asset team and the product development teams. Although each group has a unique perspective, the process instills continuity. The production plan provides the opportunity to communicate this to both the core asset and product personnel.

Large open source organizations, such as Eclipse, exhibit a similar process model. The parent organization requires a specific coordination process that places requirements on each constituent project. Within each project, there is flexibility about how development proceeds. The Eclipse organization's top level process describes how to create projects, periodically evaluate projects, and move their products to delivery [Eclipse 07]. It does not place restrictions on the day to day creation of code. In fact, the web site stresses phrases such as "just enough" with respect to process. But the

development phases they have are there for a reason, the reasons are clearly communicated, and don't require anything that is just filed away.

7 REVIEWS AND INSPECTIONS

Recently, I received a review of one of my research papers that had numerous comments all of which were of the tone, "This doesn't flow well, consider rephrasing." Another review had a single comment that said basically, "I don't believe your basic premise." The second review had a much greater impact on the eventual quality of the paper than the first. By focusing on substance that reviewer went to the heart of the matter and made a more useful contribution.

Review checklists often stress form over substance since that is easiest to see at a glance. We developed a technique that we named "Guided Inspection" as a way of focusing on substance. [McGregor 98] The referenced paper gives more details about the use of the technique.

Guided Inspection uses test cases to "guide" the inspection. We use the usual approach to creating test cases for code to create the guided inspection test cases. We start with the requirements model, perhaps a set of use cases, develop priorities among the use cases using the importance of the actors and the uses, and then use the scenarios from the use cases as the start of test scenarios. Detail is added to the scenarios to reach a level of specificity that will support a detailed examination of the work products under review.

Guided Inspection has proved to be very effective at rooting out disconnects and inconsistencies in understanding. The review sessions in which the test cases are applied to the work products are intense interactions between the reviewers and the producers of the work products. I have never left a guided inspection session feeling that the review was superficial or not productive.

8 SUMMARY

Dilbert would say all of this more concisely, and cleverly, than I have. If your latest work were submitted to Dilbert and friends, who would be more impressed? Dilbert? Or the pointy-haired manager? Correct form may get you by, but innovative substance will give you a strategic advantage.

In a rush to meet deadlines substance often is sacrificed. It is sacrificed because it is what takes the time, but it is after all where the value lies. I ask myself, before I release a paper or report, what is the substance? Will someone read this and gain information? I hope this column has at least made you consider your own output. And maybe it has given you a couple of ideas of what to do to make your efforts more effective for you and your company.

9 ACKNOWLEDGEMENTS

Thanks to Greg Barnette of Clemson University for comments that greatly improved the paper.

REFERENCES

[Bass 03] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*, Addison-Wesley, 2003.

[Eclipse 07] Eclipse, http://wiki.eclipse.org/index.php/Development_Process_2006_Revision_Final, 2007.

[McGregor 98] John D. McGregor. *The Fifty Foot Look at Analysis and Design Models*, Journal of Object-Oriented Programming, July – August 1998.

[McGregor 04] John D. McGregor. *Software Architecture*, Journal of Object Technology, vol. 3, no. 5, May-June 2004, pp. 65-77, http://www.jot.fm/issues/issue_2004_05/column7/

[RUP 07] IBM, Rational Unified Process, http://www-306.ibm.com/software/awdtools/rup/, 2007.

[Russ 00] Melissa L. Russ and John D. McGregor. *A software development process for small projects*, IEEE Software, 2000.

[SEI 07] Software Engineering Institute, http://www.sei.cmu.edu/productlines/framework.html, 2007.

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.