

Pay me now or pay me more later

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

Abstract

Institutionalizing innovation is hard work. The status quo is known, the new world is not. Managers whose primary goal is immediate risk reduction are rightfully cautious about new techniques and strategies, but change is inevitable. I want to consider an example involving software architecture definition in this issue of Strategic Software Engineering and attempt to illustrate both the new technique and the issues that are raised by adopting it. I will offer some suggestions on easing the transition.

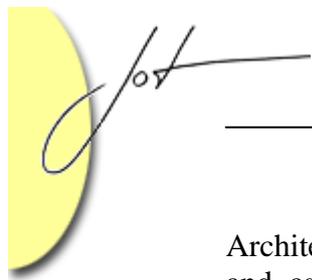
1 INTRODUCTION

All of us like the comfortable feel of routine, the feeling that we know what to expect next. This is the feeling we get with family holiday celebrations or religious ceremonies where we repeat those activities that have been successful in the past in making us feel good. In technology-dominated fields the only routine we have is constant change and yet we still have not become comfortable with change.

The architect says, “Our customers are telling us that our competition is delivering products that can be modified much more quickly than ours. I have mentioned several times how new approaches to architecture definition make it possible to have a much more specific architecture”. The manager says, “I don’t have a problem with changing how we architect a system as long as it doesn’t impact cost or schedule. What do we need to do?” The architect responds, “We can start using AADL. It’s no big deal, just a new design notation. It shouldn’t take but a couple of days for my people to learn the language.” I predict it will be a long, frustrating project. Neither the architect nor the manager is thinking through the implications of change nor is the manager being realistic about time.

In this issue of Strategic Software Engineering I want to examine a specific change, the increased use of modeling and simulation for architecture definition, both as a specific practice that is changing and more generally a change that must be managed in the organization. This change is part of the move toward model-driven development and represents a significant change for some organizations. Model driven approaches are not just a change in language or process but a change in the level of abstraction at which we do our work

The second issue of Strategic Software Engineering [McGregor 04] was about software architecture so I will not repeat the basics, but changes have occurred since that column. I will discuss a tool chain based on standards and current research.



Architecture description languages (ADL) have become more precise and expressive and, as tools mature, are gaining widespread acceptance in industry. Ideas about what constitutes an architecture and adequate architecture documentation have been refined.

In the next section I will define the context in which this discussion applies. Then I will describe the tool chain for architecture definition that blends several recent approaches to architecture modeling into an effective architecture definition process. Then I will discuss how to implement this kind of change in an existing organization, which has legacy employees and processes.

2 CONTEXT

Change implies that something, which already exists, is about to be modified in some manner. Very small incremental changes, such as new versions of a product, are routinely introduced into our business and usually little resistance is evidenced since little effort is required to accommodate these changes. Huge, disruptive changes, such as layoffs happening as the result of an economic slowdown or merger and acquisition, often are the result of factors beyond our control and are usually grudgingly accepted as unavoidable. It is the medium-sized, yet still significant, changes that result from the gradual evolution of techniques and understanding and for which there is no glaring immediate demand, that often aren't made when they should be because they will disrupt routine operation.

Many of the advances in maturing practices such as software architecture definition are gradual changes in understanding that eventually require significant changes in methods. Often some event, like feedback from customers, triggers the change to the new technique. No practice is isolated from the other elements of the software development method. When we decide to change the ripples are difficult to contain.

The software architecture community has made tremendous strides in making architecture definition a more definitive activity, but there has been no breakthrough that would cause everyone to change practice immediately. The changes have been subtle but steadily moving practice to a new level of abstraction and to a new degree of detail in the descriptions. A complex project that pushes the architecture practice at a company is often the precipitating event for management to realize a significant change is needed. In the next section I will walk through a tool chain that results in an architecture that is based on reasoning and experience and provides the increased level of detail.

3 TOOL CHAIN

The software architecture of a program is defined by the SEI as “the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them”[Bass 03]. Architecture definition serves as the bridge between requirements specification and detailed design of a solution. In this section I will focus on describing a three step tool

chain, shown in Figure 1, that supports architecture definition. Each step adds more detailed knowledge about the architecture and increases its fitness for purpose. Each step also feeds back to the previous step as the more detailed information reshapes the more general information. Then in the next section I will talk about how to manage this amount of change.

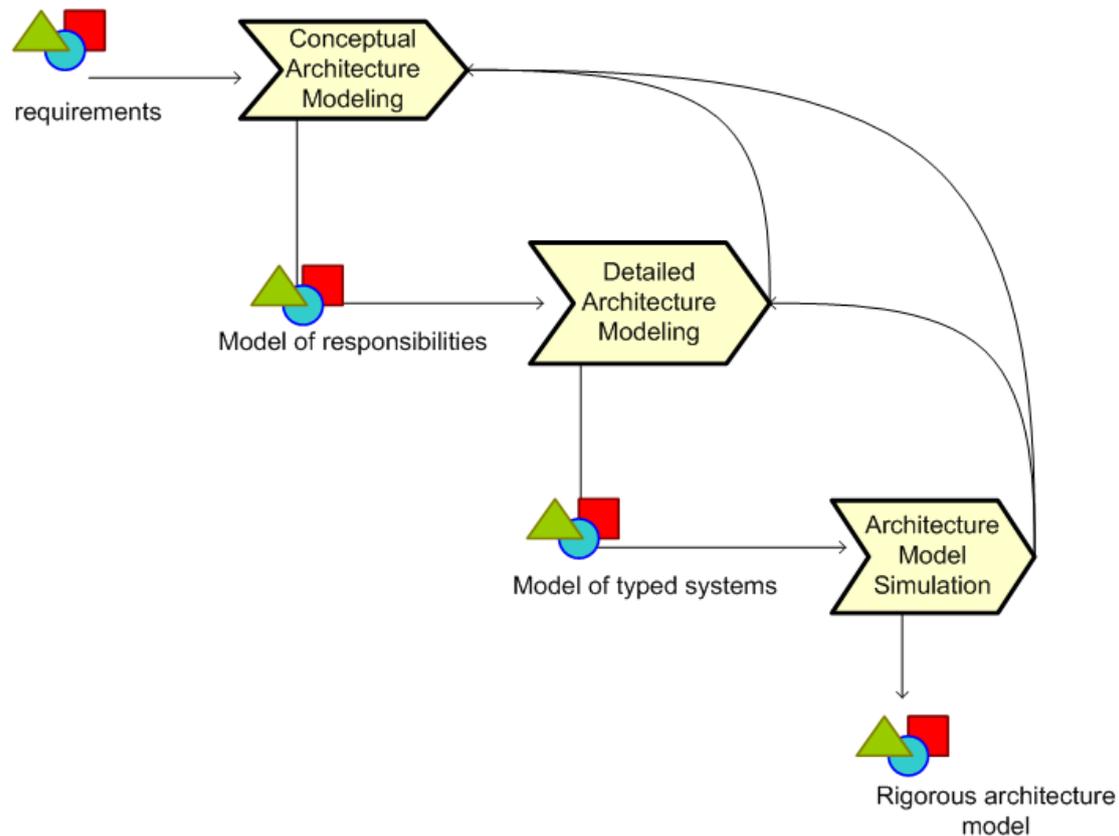
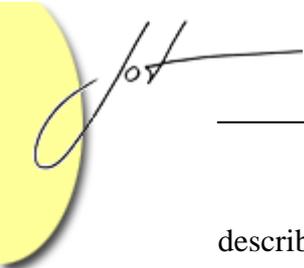


Figure 1 - Architecture Definition Tool Chain

Conceptual architecture model

The architecture is the first expression of a solution for the requirements. There needs to be a very direct mapping between the requirements and the modules in the architecture. The non-functional requirements, such as performance, maintainability, and security, should be addressed at the same time as the functional requirements.

There has not been as direct a mapping between these non-functional requirements and the architecture as with the functional requirements. The first tool in the chain provides that mapping. It takes as input a requirements model consisting of use cases containing scenarios that specify the functional and non-functional requirements of the desired system. Each use case is a direct statement of functional requirements and an indirect statement of non-functional requirements. These requirement scenarios are used to create quality attribute scenarios, direct statements of non-functional requirements. The scenarios are captured in ArchE, an architecture expert tool developed by the Software Engineering Institute[SEI 08]. Each scenario



describes a series of system actions that require a certain level of the quality attribute represented by the scenario.

The architect uses responsibility driven design [Wirfs-Brock 02] to identify the system responsibilities that are needed to realize the scenarios. Each responsibility is annotated with an estimate of the level of the quality attribute. Table 1 shows part of this mapping. Note that the quality level associated with the scenario is the **required** quality level while the value associated with the responsibility is an **estimated** quality level for the proposed responsibility. The architect identifies and models the relationships between the responsibilities resulting in a graph of responsibilities. This graph is an early expression of the architecture.

| Scenario | Required quality level | Responsibility | Estimated quality level | |
|------------|------------------------|----------------|-------------------------|--|
| Scenario 1 | .9 | Resp1 | .96 | |
| | | Resp2 | .97 | |
| | | | | |

Table 1 - Example of scenario/responsibility relationship

ArchE uses a reasoning framework defined for the particular quality to estimate the level of the target attribute, which would be present in a product developed using the architecture, given the relationships among the responsibilities and the estimated attribute values of each responsibility. ArchE makes suggestions as to which architecture tactics, when applied to the graph, are likely to move a quality attribute value closer to the required value given in the scenario [Bachmann 03]. A tactic is a modification of the graph of responsibilities based on some architecture pattern. The architect can either choose a tactic, from among those provided by ArchE, and allow ArchE to automatically apply it or the architect may make changes manually, after which the graph is reevaluated. ArchE recalculates the value of the quality for the attribute for the architecture and indicates which scenarios have been satisfied and which remain to be satisfied, shown by green and red circles, respectively, in Figure 2. After several iterations, the result is an architecture modeled as a network of responsibilities with the target levels for the various attributes achieved.

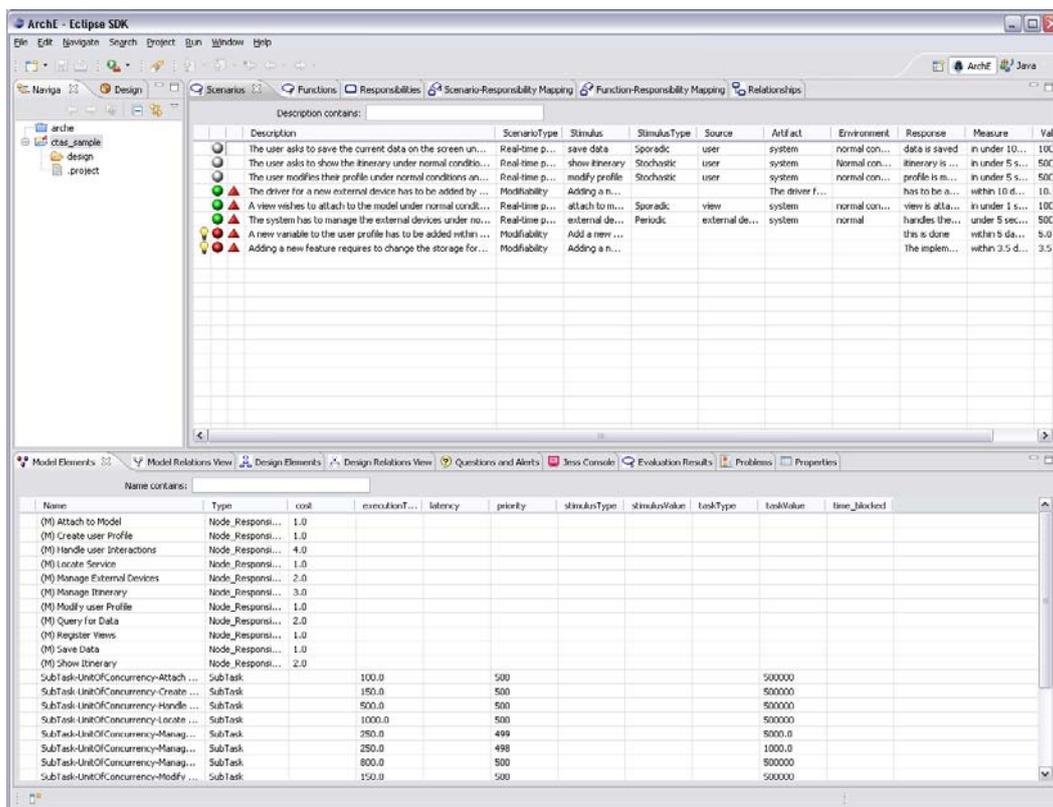


Figure 2 - ArchE

Detailed architecture model

In the second step, the model of responsibilities developed using ArchE is modeled in the Architecture Analysis and Design Language (AADL) [SAE 04] using the OSATE toolset[OSATE 08]. The graph of responsibilities becomes a graph of component specifications, shown as subcomponents in Figure 3 joined together by connections that include the relationships between responsibilities, shown as connections and flows in Figure 3. AADL has an extensible set of properties for each architecture element that can be used to capture attribute information from the ArchE graph. AADL also has a broader range of element types than ArchE so more detail can be added to the model including a characterization of the deployment environment for products developed from the architecture.

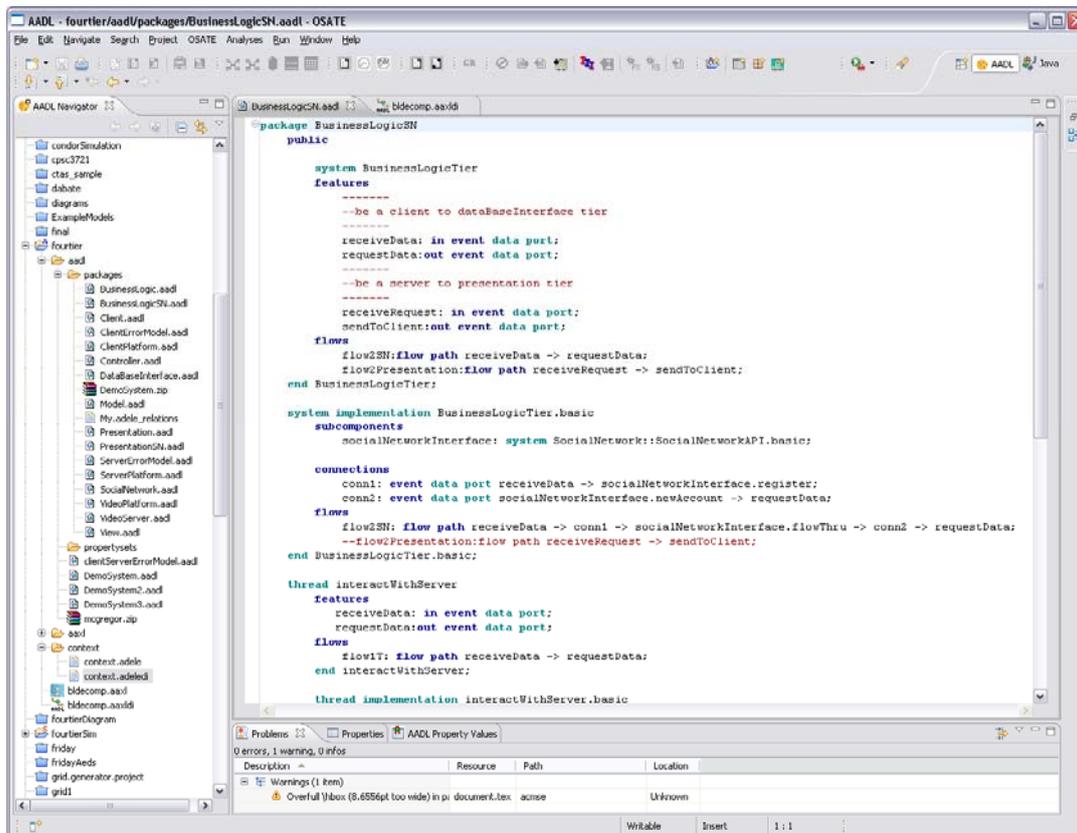


Figure 3 - OSATE toolset

AADL provides constructs for specifying both the software and the hardware to which it will be deployed. The OSATE toolset includes editors and a compiler that support the strong typing of AADL. The architect can create a detailed AADL model with numerous properties annotating each element in the architecture or a more abstract model in which only basic specifications of modules are modeled. The output, some of which is shown in Figure 3, is an architectural model that is a rigorous, detailed specification.

Architecture model analysis and simulation

In the third step, the detailed AADL model is analyzed and refined using a variety of tools. Static analysis tools address issues of definitional structure such as design complexity and flow latency. Simulators of the dynamic behavior provide the ability to examine the interactions of the elements of the architecture including load on the system.

Simulators such as ADeS [ADES 08] and Furness [Furness 08], shown in Figure 4, provide opportunities to examine the dynamic interactions of threads. The simulation runs provide the architect with an easier way to envision the dynamic behavior of the system.

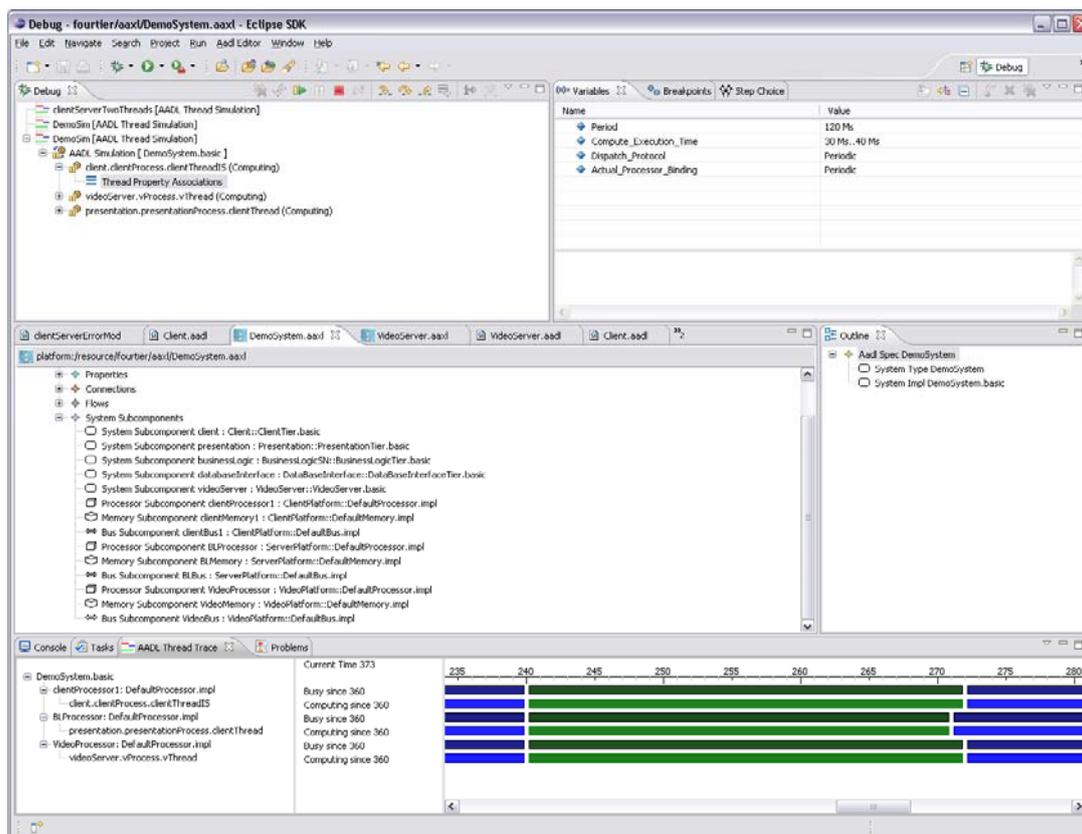


Figure 4 - Furness simulation environment

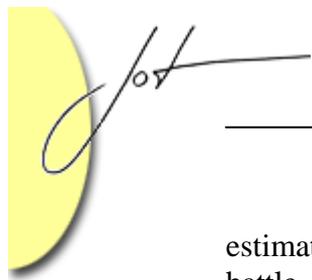
The chain

The chain described in this section represents a much more thorough approach to architecture definition than most organizations currently use. Portions of the chain can be used. For example the initial model might map requirements directly to an AADL model. Or the output from ArchE may be used as the final architecture. The reports in conferences such as WICSA and SATURN from users of parts of this approach show very positive results. However, each tool adds value so any partial chain will not provide the support of the full chain. This is certainly one of those significant changes that must be managed carefully.

4 MANAGING CHANGE

There are a number of issues regarding introducing new technology that are generally applicable. In this section I will apply these to the discussion of modifying architecture definition practice as described in section 3.

Any significant change requires a mini-business case. Any change should be considered carefully to determine its exact scope, its impact on other parts of the organization, and whether the proposed change will actually solve the problem. I do not mean a large, detailed business case. A back-of-the-envelope case is often sufficient. Its purpose is to organize decision makers' thoughts, arguments, and



estimates. In this way, costs and risks are less likely to be forgotten in the heat of battle.

The business case for most process changes is simple: Pay me now or pay me more later. Many of the significant changes that are strategically important in software engineering will payoff in terms of decreased waste and rework eventually. Not making the change will simply continue the pattern of rapidly advancing to the step where worthless code can be written, rewritten, re-written, ... you get the idea. We even have a technical term for it – refactoring. Now there certainly are situations in which new ground is being plowed and agile methods are a good match for the exploration that must happen as we gain understanding, but even in that context, refactoring a design is useful, but refactoring code is waste.

For example, why adopt the model-driven architecture definition approach? If your idea of architecture is a picture, or even pictures, then the model-driven approach is not for you. Modeling is not intended to produce presentation style graphics. Modeling captures more information than two dimensional relationships among elements. You can derive presentation graphics from the model but the model's main purpose, and hence its main payoff, is to support automated analysis, reasoning, and understanding of the system under development. Presentation graphics are for human-to-human communication and do not facilitate analysis of the system.

The business case simply states the value of the change from the perspective of customers and the organization. To determine this, the decision makers should have to consider what the change will accomplish and what value that will add to products or to processes. If all you want is a better set of diagrams, the time required to build adequate models is too high a cost. Developing the business case will encourage the thought necessary to understand these issues and planning the change will organize those issues in a coherent process.

In my experience the most glaring error in most business cases is the failure to consider the "Total cost of ownership." In particular, many of the model-driven techniques are intended to change the profile of resource allocation and to ultimately reduce the amount of resources needed. If more resources are used early in a method than a manager is accustomed to, there is the tendency to panic and halt the new activity without reaping the benefit of lower than normal resource consumption later.

Any significant change requires planning. As I have said previously [McGregor 06], the act of planning is more important than the plan. Just thinking through what needs to be done to accomplish the change, before starting the change, will identify numerous issues that if left unhandled will greatly increase the risk. Without planning there is no reasonable basis for estimating the resources required for the change. The planning feeds the business case by identifying resource needs and identifying risks.

The architecture definition technique defines a sequence of deriving one model from another that forms the basis of a plan. Each of these models carries information that its predecessor did not. Planning this change is easy because the sequence of models provides a natural progression for integrating the technique in the larger development process.

Any significant change requires a manager with responsibility for the success of the change. In fact a manager is the one who defines what success means.



Although I seldom admit this, managers do serve a purpose. An unmanaged effort will be a lower priority for the personnel carrying out the work than tasks that a manager is actively monitoring.

The manager will be working in a new environment. Historic information, metric values, and rules of thumb may not apply any longer. The manager should not make quick judgements when standard allocations of resources prove incorrect.

The manager must also get the support of all the affected stakeholders. A dictated change will be much harder to accomplish than one that has the support of the technical people whose activities will be changed.

Any significant change requires more than just new tools, it will require new skills and behaviors. For example, expecting personnel to be able to understand AADL without having time to study, maybe work through a few toy examples, is unrealistic. Even with expert outside help, personnel will still be less efficient at their job until the new method becomes the routine. Expecting personnel to change their mode of operation without sufficient incentive or clear direction is also unrealistic.

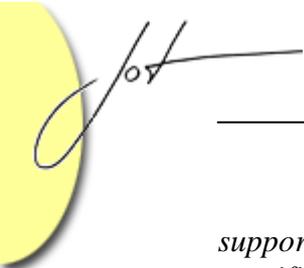
Any significant change will cause related tasks to take longer, during the learning curve, than they did during previous operation. Managers must include in their planning the extra time it will take for personnel to do tasks that they are doing in a new way. This is part of the investment in the change. Which brings the last item.

Any significant change should be viewed as an investment that will payoff later. If you need immediate savings, without the investment of time and effort, I can't help you. A significant change will cost more initially than it saves. The savings coming with time. The business case should include an analysis of the cost of not implementing the change because the event that has led to this change may actually increase costs above their traditional levels. The change may simply allow the organization to not lose ground. Part of the investment that is difficult to capture but a necessary part is to estimate the increase in value of the personnel who will receive experience with new techniques.

5 IMPLEMENTING CHANGE

The discussion in the previous section might lead you to swear off ever changing anything, but change is inevitable. You can change now under your own terms or you can change later in order to catch up with your competition. Lets examine the impact of these issues with respect to changing to the architecture definition practice described above. What would it take for this change to be successful? To answer this lets listen to the brainstorming of a manager who understands how to implement change effectively and has been given responsibility for implementing the new architecture definition practice:

I know that the scope of this change will depend on our current practice. Our current architecture definition practice involves simple box and line drawings, using colors in the boxes to denote specific attributes. The scope of our change will be large – actually huge. This is a change that adds a modeling paradigm to the software development process in addition to changing the architecture definition practice. Many of my people like to work directly with the source code. I will need to build



support for a model-driven approach to development in general and to this tool chain specifically.

I can not plan this change until I have an operational definition of the new process. I need to ask the process engineering office to assign a person to help me take the idea to the correct level. This new process will call for different numbers of people at different times than my usual approach. I need to work with my team leads to be certain that we have a realistic resource plan given this new profile.

Part of what I need to include in my plan is effort on my part to evangelize about model-driven development and this approach to architecture definition. I need to explain to the implementation manager about the new information that will be available to them eventually as the conceptual architecture is replaced by the detailed architecture definition. Essentially I am designing an experiment, like the ones in the kaizan technique of the Toyota Production System. I need to identify the data I want to collect to be able to evaluate the experiment. For example, I would like to know what percentage of the product is generated automatically.

I need to provide my architects with support during this transition. Their current idea of how to define an architecture is far from the detailed specification envisioned by the new approach. I will first get them to read some tutorials on the SEI's ADD approach to architecture definition and to examine samples of AADL specifications. They need to have a clear understanding of what has been missing from the previous architecture definition practice, what types of errors were made, and the impact these errors had on the rework of the architecture. They also need to understand how the new technique is intended to remedy these deficiencies. I will ask the design and implementation managers to provide a briefing that illustrates some of the major problems we have had late in projects because of insufficient architecture definition.

I will need to allow more time than my normal schedule would give to defining the architecture. I need to keep others moving forward so I will use the conceptual architecture as a basis for making assignments and starting the specification of interfaces. This will probably add overhead to this initial effort because they will have to later modify their work to fit the detailed architecture. This approach will reduce the pain of learning and relieve some of the pressure on the architecture team as they struggle with the new technique.

My original business case for making this change was informal but it showed that there would be a reduction in the amount of rework during implementation with an architecture developed in this way. I will probably need to make the business case more formal now to give more detailed estimates of costs and savings.

This manager has a realistic view of what it will take to make this change. He is prepared to not only change the technologies but to alter his management approach as well.

So, what is the best way to ease the pain of change? Anticipate it and plan realistically for it. Will that give you 28 hours in the day to have time to change? No, but planning will avoid some of the missteps that can occur in doing something for the first time. Being realistic ensures that if you are given permission to move ahead with a change you are less likely to have to abandon a change with the waste and embarrassment associated with that.



6 CONCLUSIONS

Change is hard. It can't be taken lightly and it can't be ignored. It has to be approached realistically and systematically. It has to be approached with an open mind about what it will take to have a successful change activity but also with a clear understanding of what will be gained eventually.

Change must be planned for and its implications must be anticipated. Part of that anticipation is managing the expectation of employees and customers and engaging them in the change process.

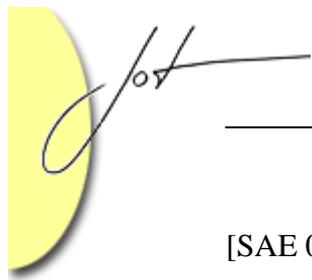
The ability to change is strategically important to an organization in our business. I have provided a small example of a typical type of change that software development organizations must make to stay relevant in our evolving world. Prepare yourself and your organization to be open to and accepting of change, but have a realistic approach to implementing those changes.

ACKNOWLEDGEMENTS

I would like to acknowledge Raj Ramlagan for his comments and suggestions that added much value to this column.

REFERENCES

- [ADES 08] <http://gforge.enseeiht.fr/projects/ades>
- [Bachmann 03] Felix Bachmann, Len Bass, and Mark Klein. Deriving Architectural Tactics: A Step Toward Methodical Architectural Design, Software Engineering Institute, CMU/SEI-2003-TR-004, 2003.
- [Bass 03] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*, Addison-Wesley, 2003.
- [Furness 08] <http://www.furnesstoolset.com>.
- [Jones 08] John Jones, DeAnne Aquirre, and Matthew Calderone. 10 Principles of Change Management, Resilience Reports, <http://www.strategy-business.com/resilience/rr00006>.
- [McGregor 04] John D. McGregor: "Software Architecture", in *Journal of Object Technology*, vol. 3, no. 5, May-June 2004, pp. 65-77. http://www.jot.fm/issues/issue_2004_05/column7/
- [McGregor 06] John D. McGregor: "Planning before plans", in *Journal of Object Technology*, vol. 5, no. 2, March-April 2006, pp. 27-34 http://www.jot.fm/issues/issue_2006_03/column3/
- [OSATE 08] <http://www.aadl.info/>.



- [SAE 04] Society of Automotive Engineers (SAE). Architecture Analysis & Design Language (AADL), SAE Standard AS-5506, 2004.
- [SEI 08] Software Engineering Institute,
<http://www.sei.cmu.edu/architecture/arche.html>.
- [Wirfs-Brock 02] Rebecca Wirfs-Brock and Alan McKean. Object Design: Roles, Responsibilities, and Collaborations, Addison-Wesley, 2002.

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-based software engineering. His latest book is *A Practical Guide to Testing Object-Oriented Software* (Addison-Wesley 2001). Contact him at johnmc@lumsoft.com.