To be published in the ICSE and INCOSE 2004 Region II Conference (9/15/04) Proceedings

Migrating from a Waterfall Systems Engineering Approach to an Object Oriented Approach – Lessons Learned

Robert J. Cloutier
Lockheed Martin Corporation
Stevens Institute of Technology

Abstract

A large number of the system of systems developed over the past decade have been software intensive systems. The predominant systems engineering approach in use over that period is the waterfall process, sometimes referred to as a 'V' methodology or model. In this model, steps that are defined as the sound engineering rigor are executed in a sequential manner. This process has been in use for over 20 years by most Department of Defense contractors (as well as commercial enterprises that employ a systems engineering discipline). However, the past decade has seen a migration to object oriented (OO) methodologies for software intensive systems. This methodology typically brings other technologies, products & processes which impact the way organizations develop systems. This paper addresses lessons learned from both a project manager's perspective and an architect's perspective in producing large system of systems using an object oriented systems engineering methodology.

1 Overview

The transition from the traditional waterfall development to an Object Oriented (OO) approach can be stressful for any organization. The transition is occurring with the acceptance of OO methodologies for most types of applications, including real time systems. For instance, current Navy Open Architecture guidance states, "The Object-Oriented Programming (OOP) method did not originate in the real-time community, but it has matured to the point where the performance characteristics of OOP computer programs are, in some cases, adequate to meet real-time requirements. The OOP approach not only supports critical OA Objectives, such as affordability and time-to-market, but also represents state-of-the-practices among trained engineers within the pool of likely OA implementers. Where OOP technology does not support real-time requirements, its use should be avoided."[1]. The OO methodology challenges the documented organization processes and experiences, and may cause the experts to feel like beginners again. It introduces new artifacts along the way that are unfamiliar to waterfall-oriented management. Below is a table of terms that are used to compare the two approaches. There is not an exact one for one relationship. However, it represents a very common comparison of concepts between the waterfall approach and object oriented (OO) approach. It also represents the technologies that were implemented on the projects representing the lessons learned herein. Each of the concepts listed under the OO approach will be discussed briefly before moving onto the lessons learned.

Waterfall Approach	OO Approach
Waterfall methodology	Iterative development and delivery of new products
Structured analysis and functional decomposition	Object-oriented analysis and design for systems engineering
Stand-alone tools and diagrams	Model driven architecture of the entire system and subsystems
Structured programming	Component based software development

Table 1 - Comparison of Concepts & Terminologies

1.1 Iterative development

The waterfall methodology or approach is used by many, if not most large organizations today. It appears to have been first put forth by Royce [2] in 1970. Figure 1 is a representation of the waterfall approach. This system development approach is characterized by cascading phases from systems requirements to operations and maintenance. The model is implemented in a number of different variations, including the most formal approach, where it is necessary to complete and formally review the artifacts created in one phase before approval is granted to move to the next phase. For a large project that spans many years, some of the phases can be 6 – 18 months long.

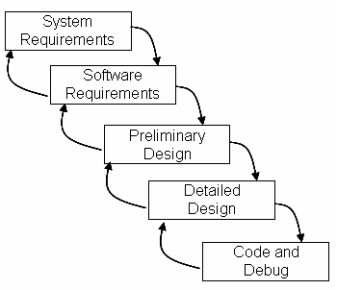

Figure 1 - "Waterfall" Development Model

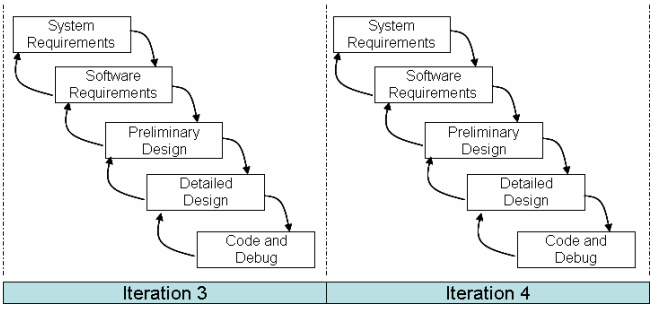

Figure 2 - Iterative Development

Iterative development, made popular by IBM Rational and their Rational Unified Process (RUP), implements this cascading approach into smaller iterations. Iterations become complete development waterfalls with much shorter durations. For instance, the current project the author is working on has structured the program with 3-4 month iterations. If the program is a multi-year program, iterations can be grouped into spirals, and incrementally improved products can be delivered at the end of each spiral, as suggested by Ericsson [3]. An example of this type of scheduling would be a 36 month program structured with two 18 month spirals. Each spiral could have 6 iterations with durations of 2 – 3.5 months each. A portion of this concept is represented in Figure 2.

1.2 Object oriented (OO) systems engineering

OO systems engineering has been documented by other authors, to include Friedenthal [4] and Cantor [5] [6], and is even recognized in the INCOSE Systems Engineering Handbook [7]. To quote that handbook, “In object oriented modeling, the fundamental construct is an object which combines both data structure and behavior in a single entity to represent the components of a system. Other fundamental characteristics include classification, or the grouping of objects with similar data structure and behavior; inheritance or the sharing of properties and behavior among classes based on a hierarchical relationship and polymorphism in which the same operation may behave differently on different classes of objects.”

1.3 Model Driven Architecture (MDA)

Model driven architecture, from my experience, is the development of architecture models using a modeling language, such as UML or the proposed OMG SysML. On the Open Management Group’s website, you will find that they define MDA to encompass considerably broader concepts - including the use of other open standards to include XML, XMI and others. For the purposes of this paper, it is enough to know the lessons learned here apply to projects which used UML to design and document the system architecture, and then flow those designs and attributes down to the subsystems.

1.4 Component based software development

Component based software is the movement from large, monolithic software programs to modular software structures that are more manageable, scalable and potentially reusable. Maintainability is improved with the ability to update a component instead of a large software upgrade. Components may or may not be object oriented, though the OO community has truly embraced component based software development. Examples of commercially available components can be

found in the aftermarket grid components that are available for Microsoft Excel. Most of the "plug-ins" to Microsoft Windows based software are nothing more than consumer installed and configured components

2 2.0 Lessons Learned

As mentioned in the overview, most of the methodologies discussed earlier can be implemented individually, and may cause an organization pause. However, the projects in which I have been involved have chosen to implement them all at once on a single project. It is not the intent of this paper to attempt to justify one methodology or another, but rather to anecdotally relate the lessons learned making that transition to the collection of methodologies referred to as the OO approach. Finally, there is no implied significance to the order in which the lessons learned are presented.

2.1 Document your processes as soon as possible

Formal process tailoring to receive a level 3 or higher CMMI tailoring is a challenge in anyone's book. This author has participated in SEI/CMM or CMMI certification twice and Systems Engineering process certification once and it appears that the standard processes recommended by those certifications favor a sequential, waterfall process. If your organization has a process group whose experience is waterfall based instead of the more modern approaches, there will have to be a significant education process that must be accomplished as part of the tailoring process. This is necessary so the process management organization will approve the processes necessary to perform iterative, OO development (or Agile, or any other modern development methodology). If the plan is to begin the process tailoring the same time the project is starting, the project manager may find the team is halfway through the project and still not have organizationally approved processes for the team. And, without the processes, the QA organization will have no idea what to perform process audits against.

Examples of why this is important are easy to demonstrate. First, few organizations that are transitioning from traditional systems engineering approaches will have UML guidelines in place. Without them, the use case specifications, activity diagrams, sequence diagrams, etc. developed will have an inconsistent look and feel from engineer to engineer. The second example of the need for documented processes early might be common coding guidance for OO programming languages. If those aren't defined early, the project team may be developing prototypes without coding guidance. Another example is modeling. A final word of advice here - whatever your project team believes is a credible estimate of effort to sufficiently define and document the processes, and then gain consensus and approval, take the estimate and double it!

2.2 When adopting a new methodology, do not underestimate the value of team buy-in

Unless an individual is a change agent by nature, the initial tendency is to resist change and try to adapt the way they are comfortable into the new approach. One example of this is when it comes to implementing the use of use cases instead of traditional specifications. The first time most engineers author a use case, it looks more like a specification than a use case, even after being trained on how to write use cases. The Rational Unified Process™ (RUP) is a methodology from IBM Rational that is often associated with OO projects. The initial reaction to moving to RUP is to implement the entire methodology instead of only using the parts that make sense for the project at hand even when there may be members on the team who are experienced with RUP. Figure 3 is a diagram to help explain this phenomenon.

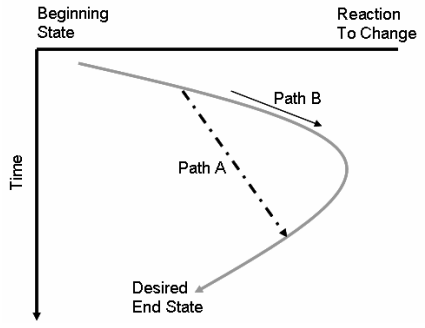

Figure 3 - Learning Curve

When a change is introduced, the team is launched down a learning trajectory. The obvious goal is to help the team take a shorter route, Path A, to the desired end state of fully transitioned to the new methodology. The lesson learned is that the team will almost never take Path A. They will stay on the learning curve, overshoot by "following the book" and then come back to something more acceptable by way of process improvement. We finally came to the conclusion that instead of trying to get the new team to take Path A, it is better to simply help them get down Path B faster, arriving at the same desired state in the same timeframe. Team members had to make their own mistakes, and discover the corrections needed to devise an acceptable methodology.

2.3 Create and maintain a master project schedule

The iterative philosophy is to create a risk based schedule, placing the high risk and architecturally significant tasks

early in the project plan, and then begin designing and prototyping - build a little, test a little. As the project moves from iteration to iteration, adjust the schedule as necessary. Therefore, the project begins with a project plan that has a high degree of fidelity in the close in months, and a coarser granularity in the out months. That is contrasted to the waterfall method which results in a complete project plan at the start of the project, planned to high fidelity of detail through the product completion and shutdown.

The lesson learned is that the iterative approach of scheduling works fine on smaller projects and e-Commerce projects. However, for the large, multi-year programs that are common in DoD, there still has to be a full blown, integrated master schedule, sometimes referred to as an IMS. I have seen at least one lesson learned paper that suggests this is the wrong approach. The argument goes like this – it takes away the flexibility gained by the iterative process. However, without a sufficient level of detailed planning, no single individual can keep all the program complexities and dependencies on track without a complex plan. At a minimum, have a rolling, detailed plan for the next year to 18 months, and a less detailed plan which contains significant milestones and dependencies for the remainder of the project.

2.4 Train your team early and train them often (and make sure there is adequate funding)

Traditional functional systems engineering utilizes functional decomposition to organize the system into smaller, more manageable subsystems, and then to allocate the system requirements to those subsystems. This decomposition becomes the system architecture. Once the subsystems are derived, the requirements for those subsystems are derived. The INCOSE Systems Engineering handbook [7] defines the object oriented systems engineering methodology this way: *"…it defines the system as a collection of interacting objects that need to work together to provide the expected solution to the defined problem and maintains that perspective throughout the development lifecycle"*. The OO approach requires training to perform it properly. If the team is asked to perform the object discovery and definition without the proper training, the result will be as ineffective as would someone unfamiliar with functional decomposition being asked to perform that task without the requisite level of training.

Peter Senge emphasized the value of the learning organization [8] in 1990. An organization must be willing to empower their teams to continue learning. This seems to be a self evident truth, yet it is interesting how difficult it is to get some management organizations to spend the money on training. We do not hire high school graduates and expect them to become engineers by providing four years of on the job training. Why expect teams to learn a new systems development methodology without the proper training? The lesson learned is that training early and often will cost more early on in the project, but it will pay large dividends on the back end with high customer satisfaction because the project was completed on time and with good results.

A corollary to this lesson learned is that once trained, team members must be provided adequate mentoring to develop an experience base on which to build. Don't expect newly trained individuals to return from training ready to be productive. The examples normally used in class bear very little resemblance to any real world domain in which they will be asked to perform, and they will require time to assimilate newly learned skills into the actual working domain.

2.5 Enforce architecture integrity across the subsystems

As mentioned earlier, when using the OO approach, UML is many times the modeling notation of choice. One of the advantages, and disadvantages, of UML is that it can be applied in numerous ways to represent a system. This can lead to models that are inconsistent. For instance, one systems engineer may want to develop complete, text based use cases, while another may favor the combination of graphical use cases and activity diagrams. Though some would disagree, this may simply be a matter of style. However, when requirements traceability is required, the differences in modeling approaches will make that task much more difficult. The same is true of architecture elements. There needs to be a chief architect, or an architecture team to ensure that top level architecture elements and patterns are carried through the design, and applied properly at all levels. Another chief role the architect team must play is communications channel. When one team on the project implement an architecture construct that may be useful to another team on the project, it is the architecture teams responsibility to communicate that direction. This will improve consistency of the models and reusability of the code for software intensive systems. Finally, consistent with the prior lesson learned, training on the architecture is also important. Once architectural decisions are made by the system architect, the teams working on the subsystems must be trained on those decisions.

A corollary to the architecture integrity lesson learned is that architecture prototyping is critical. Encourage the system architect or architecture team to construct a prototype of the architecture. If the team has done any performance modeling, use the prototype to confirm the expected performance. If the architecture is sound, the prototype may become

the beginnings of the infrastructure necessary for one or more subsystems to build upon. If there is a problem with the architecture, risk has been avoided, and it can be re-architected without a lot of rework by the subsystems.

2.6 Conduct customer demonstrations of progress

Customers like to see progress. The waterfall process, especially when applied in the strictest form, may cause the customer to wait for many months, sometimes years before they see some physical manifestation of their vision. Even for small software projects, it may take a month or two to get all the requirements documented thoroughly, and approved by the customer. Then, the high level designs and detailed designs must be completed before beginning the coding. The object oriented approach (actually, the iterative development part of that grouping) encourages an approach that calls for requirements gathering, design and prototype code of high risk and/or architecturally significant portions of the project first. When the prototype is working and meeting the original set of requirements, share it with the customer. Ensure that what is built satisfies the original set of requirements laid out for this first iteration. This will provide early feedback to the project team on the context and direction being taken, and allow for earlier (and therefore less expensive) corrections if they are needed.

Once the architecture and high risk items are mastered and demonstrated, the next iteration begins with the gathering or maturing of the requirements for the next phase of functionality to be added to the prototype, beginning to transition the prototype into a real product. Once that is complete, demonstrate it for the customer again. The lesson is to keep the customer involved and excited about the progress with a continual buy-in process.

2.7 Find QA and CM change agents, and bring them onto the team early

Just as in the waterfall approach, object oriented systems engineering still requires the Configuration Management (CM) and Quality Assurance (QA) disciplines to effectively manage a large complex project. However, the model based architecture and model based design methodology bring new development tools, and new artifacts for these two organizations to CM and QA. Traditionally, these two organizations are brought onto the team later in the development lifecycle. The OO systems engineering approach requires these organizations become involved considerably earlier, and they have some challenging work ahead to update and document the new processes that will be necessary. Therefore, virtually all of the prior lessons learned apply to these organizations, too – documenting processes, buy-in, master schedule milestones, and training. If the CM and QA organizations do not identify individuals that are receptive to the migration, the effort to make the necessary CM and QA process changes will also be significant. The unique organizational alignment of these two organizations - particularly QA, which normally has completely different reporting structures outside the engineering department - will make it difficult for the engineering department, or the program office to dictate the change. A change agent, which can help work the politics within the CM or QA organization, can be a true project asset when migrating from waterfall methodology to an OO methodology.

3 Summary

This paper has presented seven lessons learned, based on experience gained when migrating teams from a waterfall methodology of systems engineering to an object oriented systems engineering methodology. This experience was gained from large projects, both in the commercial and DoD industries, and should apply to both types of businesses. The seven lessons learned are:

- Document your processes as soon as possible
- When adopting a new methodology, teams will buy in at their own pace
- Create and maintain a master project schedule
- Train your team early and train them often
- Enforce architecture integrity across the subsystems
- Conduct customer demonstrations of progress
- Find QA and CM change agents, and bring them onto the team early

Migrating to an OO systems engineering approach is not a wholesale rejection or replacement for traditional practices. In fact, in practice they seem to be more similar than they are different. Both methodologies require discipline, rigor, conduct of trade studies, detailed documentation, reviews, two-way requirements traceability throughout the system, and customer acceptance of the final product. The order of execution is different however, and though not discussed in this

paper, the level of maturity of an artifact before proceeding to the next phase, differs. Some of the artifacts look different, some have different names, and some are altogether new. But the goal is always the same – to produce a high quality product that meets the needs of the customer and user.

Finally, if the lessons learned presented may not appear to the reader to be object oriented, it is because they are not. In actuality, the technology takes care of itself. It is the project management and resources management issues that are the tougher challenges in migrating from a waterfall systems engineering approach to an object oriented approach. It is the resistance to change, the "not invented here" mindsets, and the documented organizational processes that become the real challenges, and even obstacles, to migrating.

4 References

[1] *Open Architecture Computing Environment Design Guidance, Version 1.0 (Interim).* 10 March 2003. Naval Surface Warfare Center Dahlgren Division (NSWCDD). Available at: http://www.fbodaily.com/archive/2003/04-April/18-Apr-2003/FBO-00305229.htm.

[2] Royce, Winston. *Managing the Development of Large Software Systems. Proceedings of IEEE WESCON* (August 1970), pp.1-9.

[3] Ericcson, Maria. *Developing Large Scale Ssytems Using the Rational Unified Process*. IBM Rational Whitepaper.Available at: http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/sis.pdf.

[4] Friedenthal, S. et al. *Adapting UML for an Object Oriented Systems Engineering Method.* Proceedings of the 2000 INCOSE Symposium.

[5] Cantor, Murray. *Rational Unified Process for Systems Engineering*. The Rational Edge. August 2003. Available at http://www-106.ibm.com/developerworks/rational/library/content/RationalEdge/aug03/f_rupse_mc.pdf.

[6] Cantor, Murray. *Rational Unified Process for Systems Engineering, Part II: System Architecture*. The Rational Edge. September 2003. Available at: http://www-106.ibm.com/developerworks/rational/library/content/RationalEdge/sep03/m_systemarch_mc.pdf.

[7] *International Council on Systems Engineering. INCOSE Systems Engineering Handbook, A "What to" Guide for all SE Practitioners*. 1 June 2004. INCOSE-TP-2003-016-02, Version 2a.

[8] Senge, Peter. *The Fifth Discipline. The Art and Practice of the Learning Organization.* 1990. New York: Currency Doubleday.