A Holistic Approach to Embedded Systems Development

Thomas A. Bullinger
President
ArchSynergy, Ltd.
Victor, NY 14564

Sandeep Mitra
Associate Professor
SUNY Brockport
Brockport, NY 14420

Abstract

The Software Engineering Effectiveness Model (SEEM™), was well received at the 2003 ESC(W) Conference. Based on feedback received, this paper reviews the latest status of SEEM, including updates to key features and introduces the latest enhancements. These enhancements include Concept Selection and the Learning-Principle-Concept (LPC) Loop. This paper and seminar follow a case study from start to finish providing a survey of practical techniques for the application of SEEM to an embedded system product.

Introduction

Rather than provide another dry, academic tome of information consistent with our paper as presented last year, we decided to try a different approach. The following paragraphs are extracted from the journal of a fictional character engaged in developing a fictional product. As the story unfolds, the elements and activities of a SEEM project become clear. Further, the benefits of SEEM are placed in suitable context providing for a more focused understanding of the material. Finally, we hope this is a far easier paper to read, and is therefore more memorable. Thanks for indulging our creative spirit!

Day 1

I was sitting at my desk this morning reviewing the results of our latest project. Once again, the Standish Group [1] study was well represented. Our project was late, over budget, and still does not meet the needs of the client who is actually going to pay for it. The bug reports continue to flood in to our help desk, and the developers are working 50 to 60 hours a week. Despite all the hours invested, the number of bugs appears to be increasing. I don't know how we can continue to support the current project and start work on the next one in the pipeline. I can't help thinking that there must be a better way.

Day 2

My boss, Sue, called me on the carpet after lunch, demanding to know why progress was so slow. I explained about the vague requirements from the customer. I discussed the immature technology that we applied. I told her the staff skill set just wasn't up to the task. I complained about the crappy tools that don't meet the expectations set by their vendors. I vilified the customer, who keeps changing their mind about the features we did implement, saying we don't understand their problem so our solution doesn't solve it for them.

My boss, being the great person that she is, asked if there was a better way to develop software for embedded systems. She explained that another product development activity was about to kick off for a major client. Poor though the performance of my project was, it was still the best results the company had seen thus far for a software project. She tells me that I have great potential as a manager, and that she knows I have a better vision for developing software. Humbled beyond words, I could only agree.

She promoted my best developer to take over my existing project, and said I could have two of my current people, and hire more from the outside as needed.

Then she explained that I could make use of any process or methods I wished, but another project with the same results as the current project would mean the end of my career at that company, and at any other company where she had friends. Failure was not an option. Her grin did nothing to soften the words.

Day 3

Contemplating my new assignment, I remembered one of the new hires brought in from RIT[1]. I remember these hires complaining about our lack of process, saying they had a far better experience developing software using the SEEM methodology. He said we were in the software dark ages, and had the completely wrong philosophy for developing software. I was irritated at the time, but thinking back I was more hurt by the truth of the statement than anything else. Out of the mouths of babes…

I called Paul into my office this afternoon, and asked about SEEM. We talked for most of the day, exploring the various aspects of SEEM and whether it would apply to the next project or not. Paul informed me that SEEM is primarily focused on figuring out what the problem is, rather than focused on a particular solution. Once the problem is well understood, obtaining the correct solution is far easier. This has the additional benefit of ensuring the customer is well represented, and their real-world problem addressed and solved. Furthermore, he went on, SEEM addresses the needs of all the other stakeholders as well, ensuring that the entire project is balanced to meet the needs of all interested parties.

Paul explained the SEEM philosophy of Triaxial Architecture. SEEM suggests that involving all the elements of a development enterprise in the project yields a better solution. Traditionally, the technology axis (engineers!) drive a solution with only limited input from the business axis (management!) and the market axis (marketing, sales and the customer!) By considering the needs of all three, and involving all three in the project wherever possible, the project gains better focus and improved productivity. I asked how SEEM manages to get the three groups, which typically only meet at parties, to talk together. Paul explained that SEEM creates a project at multiple levels of abstraction, and that the higher layers are suitable for non-techies to read and understand. By providing simple tools to communicate thoughts and ideas, it is easier to discuss the issues with all the stakeholders. Paul asked if I ever reviewed a DFD[2] with a customer. We had a good laugh over that one!

[1] Rochester Institute of Technology
[2] Data Flow Diagram

I asked Paul if we had to invest in pricey CASE tools, and deal with their proprietary interfaces, to make the best use of SEEM. He explained that the SEEM philosophy, as drawn from eXtreme Programming [2], is to keep things as simple as possible. To that end, only well-known tools such as Microsoft Visio, Word and Excel were used to create the documentation artifacts[3]. He did say that the development environment is still needed for writing, debugging and executing code, but we already had suitable (and free!) toolsets for that.

I told Paul that SEEM sounded like a great methodology, but wondered where it came from. I'd heard of eXtreme Programming (XP) and Rational Unified Process (RUP), but not SEEM. He explained that SEEM was essentially an Agile Methodology [3], built on the best practices of both XP and RUP, and incorporating their respective philosophies into it. The developers of SEEM have been practitioners for years and years, and know how to get projects out the door on time. SEEM has been developed to such a fine degree, that they have a "recipe" for developing software, something no other methodology has ever provided. SEEM claims that if you follow their recipe, and apply their metrics and heuristics properly, you cannot fail. To support this claim, Paul told me that to date, no SEEM-based project has ever been delivered late. Furthermore, no SEEM project has ever failed to meet the needs of all the stakeholders, and especially the needs of the customers that pay for it.

I asked if SEEM applied to embedded systems. He told me that SEEM philosophy actually applies to any engineering discipline, but was currently geared toward software. He assured me it would work great for our application.

I told Paul I was willing to give SEEM a try, and asked how to get started. He suggested we bring over Tim, a fellow student of his that took the same SEEM course he did. He also suggested we bring in a SEEM consultant to help them get started. Once geared up, we could hire more people with exactly the right skill sets.

Day 5

Tom, our SEEM consultant joined us today. We licensed the SEEM cookbook from ArchSynergy, and went through an introductory session on SEEM. The introductory session covered a lot of ground mostly on the philosophy of SEEM. Here are some of the things I learned:

- SEEM is based on XP, RUP and a few other methods, but has refined the techniques and philosophy to make them actionable.
- SEEM considers all aspects of the development enterprise, and in particular the business, market and technical communities.
- SEEM is focused on understanding the problem to be solved, and ensuring that the eventual solution is traceable in full back to the original problem.
- SEEM creates models geared toward communicating with people, not computers.
- SEEM discovers a metaphor for the project, and leverages the metaphor to facilitate communication.

[3] An artifact is any document, diagram or file produced during the course of a project.

- SEEM recognizes that patterns occur at multiple layers of abstraction, and leverages the patterns from one layer to the next.
- SEEM renders the actual implementation of the code trivial, as all the "hard" stuff is figured out in advance.

The team received this very well, as they already had exposure to the concepts. My boss, who joined us for the overview session, really liked the idea of focusing on the problem to be solved. She also liked the concept of full traceability, which ensures the right code is implemented at the right time.

Day 8

Paul, Tim and I have spent the past few days working out the business context diagram and stakeholder profiles. Working from a product concept provided by my boss, we touched base with many of the stakeholders to determine what they really wanted in their new application. Since we aren't allowed to talk to the actual customers (something about us being geeks…), we spent time with the marketing and sales folks. I realized for the first time that a lot of different people are involved in creating a software application and they all have their own requirements. I thought only the customer drove requirements! It's really nice that SEEM provides a tool for finding stakeholders and capturing their issues.

We spent a bit of time with the hardware folks as well, trying to figure out how to understand and capture their thoughts and ideas. We didn't understand all of it, but we captured everything and hoped we could make sense of it later on.

We also gathered information on how the product will work when completed. At this point, everyone still has different ideas about what it should and should not do, and many of them are contradictory. We didn't pass judgment - we just gathered the information.

Day 9

We spent quite some time today figuring out the right metaphor for our product. This was another first for me, as I had never considered a metaphor as a communication tool for developing software. We talked through a variety of possibilities, and in the end settled on a metaphor that fits reasonably well.

The benefits of using a metaphor were immediately apparent. Suddenly we had a vocabulary that applied to our problem, and we were able to accelerate our communication just by referring to the metaphor. We were also able to intuit more requirements from the metaphor. In our case, the metaphor is well known, and a mature model. So drawing requirements made sense as we could build on the expertise already applied to the metaphor. Even though it wasn't a one-to-one correspondence, the translation from the metaphor domain to our problem domain was easy.

Tom tells us that metaphors will apply in different ways throughout the process, showing up in the form of a similar problem, or the form of a problem-domain pattern, or even as a design pattern or coding idiom. If true, it'll be interesting to see these develop.

Day 10

We met with the senior management team today, and presented our results thus far. We needed to meet with them to gather information on the business aspects of the product development. Prompted by the SEEM templates, we asked them a large variety of questions, some of which took them off-guard. For example, when we asked them the mission of the business, they were astonished that we didn't know! A secretary was called, and he quickly provided us with a corporate overview that described the mission and vision of the business, as well as a set of principles that guide the decision making process. We suspect this will be immensely helpful later on.

We also talked about the roadmap for the product, even though we have only limited requirements. Rather than attempt to produce the be-all end-all product of all time, we put together a strategy for rolling out the features over time. By starting with the simplest possible product, we can deliver to the customer early, and get feedback from the use of the product. This will make it easier to evolve the product over time and ensures we really understand the problem we're solving. Although management wasn't entirely comfortable releasing a minimal product to begin with, they really liked the aggressive delivery schedule allowed by this strategy.

Along with the roadmap, we discussed a high-level schedule, or product timeline. As we knew more people were needed, we included them in the resource-planning schedule. We also described each of the project phases as suggested by SEEM, and discussed how they fit into the product release process already in place. Senior management, seeing things presented that they could relate to, provided some excellent suggestions that we incorporated into the development process.

At the end of the meeting, we all agreed that this was the most productive management meeting any of us had ever been to. This was the first of many testaments to SEEM's ability to focus activities on the right issues.

Day 16

We completed the first cut of the User Story diagrams today. This was my first experience modeling with UML's use case notation, and proved to be very effective at capturing the behavioral aspects of an application. I've always worked with textual requirements, but they tend to be ambiguous. We've all seen the mountains of verbose statements saying what shall be, what should be, and what may be, but I've always found them to be a pain. How do you translate those speculative words into software? How do you resolve the contradictions embedded in your interpretation? And worst of all, how do you determine if you have indeed met the "requirements"?

SEEM suggests the details of a particular user story to be described in text. However, SEEM provides a set of templates for describing user stories, in which the stakeholders interested in each user story identifies the preconditions, the flow of events required to achieve the results of the user story, and the measurable results. By focusing on the structure of the template, and working within the context of the user story, the results were less ambiguous. SEEM also uses UML notation to depict the relationship between the individual User Stories (which story depends on which other story, which one is a generalization of the other, and so on).

The user stories introduced by XP, and honed by SEEM are a nice way of capturing the reality of an application. They transform the ambiguity of text into the reality of a problem to be solved. Furthermore, the user stories are easier to categorize and review, and the stakeholders actually understand them!

This was also our first chance to apply SEEM in-process metrics to our work products. For example, our first diagram had 17 different user stories represented in the top-most diagram. This violated the SEEM 7+/- 2[4] rule for diagrams, but we weren't sure how to reduce the number.

Tom demonstrated how abstractive decomposition could be used to build a hierarchy of user stories while still presenting the appropriate information. While we had heard about abstractive decomposition in the introductory session, I must admit that I didn't understand it. Now seeing it applied, it makes perfect sense. It's just a matter of combining like user stories together and giving the abstract concept the right name. As long as the user stories grouped beneath the abstract user story are consistent with the name, then the abstraction is appropriate. Tom again explained the difference between abstractive decomposition and functional decomposition, and why abstractive decomposition was a better option. I think I'll remember it this time.

While working out the behavioral aspects of the system, we ran into many requirements that were non-functional. For example, the customer wants this particular product to run on a PowerPC chip. The non-functional requirements were captured in a separate text document.

Day 18

Today we received a lecture on testing. Historically, we create our test cases last, and frankly, we were never very thorough about it. We just ran our application as if we were a user, and noted things that didn't work. Of course, the users discovered far more bugs than we did, a constant source of embarrassment for us.

SEEM has adopted the XP philosophy on testing, suggesting that we consider test cases even before we start implementation. SEEM recommends two levels of testing: Customer Acceptance Testing, and Engineering Testing. Since Customer Acceptance testing is based on the user stories, this was the appropriate time to create them. I guess tomorrow we'll spend some time considering the user stories and how to test them in the final application. We'll deal with the Engineering Tests later on when the design is underway.

Day 20

We spent some time today talking about mapping user stories into sequence diagrams. Up to this point, we kept trying to sneak solution concepts into the analysis. Finally, Tim pointed out that the analysis should proceed as if we were designing a system for a bunch of people with pencil and paper. He convinced us that if our analysis could be modeled with this simple concept in mind, then we could apply any technology to our actual solution.

[4] Studies have shown that the limit of human cognition is roughly 7 items, plus or minus 2.

At first, it seemed silly to examine everything in this light, especially considering the fact that we're developing an embedded, real-time control system. Eventually, we realized that the task didn't have to be practical for people with pencil and paper, merely possible. Once we rationalized that thought process, it became second nature.

Day 23

We're mid-way though the sequence diagrams now. I was concerned that the functional nature of a user story wouldn't translate into an object-oriented system. I was pleasantly surprised when Tom indicated we were right on track. The trick to transcribing the user stories into an object model is through the correct identification of the objects for the sequence diagrams. Thinking back to my early OO days, I remembered rooting out objects by picking nouns from the problem statement and problem domain. I was concerned about some choices, but as long as we stuck to the nouns, and could rationally justify our selection, they are OK.

Again, SEEM provides guidance in this area by providing some qualitative metrics that can be applied. For example, Tom explained the Darwin test to us. The Darwin test is simply the concept of "survival of the fittest." If an object choice cannot be successfully moved into a different environment, it's probably not a good candidate for an object.

At one point, we had a serial interface object that understood a particular protocol for communicating on a serial port. After considering the Darwin test, we realized the object we chose was specific to the details of this particular protocol, and therefore failed the test. We re-factored the object into a protocol object, and a serial port object. In this way, we could reuse the serial port in any place requiring a serial port. We could also reuse the protocol in any other application that required similar semantics in the interface. By breaking the object into more natural entities, we increased the cohesion[5] of each object, and reduced the coupling[6] to other objects.

Day 25

We continue to make incredible progress on the project. The intellectual tools provided by SEEM have focused the team in a manner I've not experienced before. Now that we're most of the way through the analysis, we've seen all the UML diagrams and templates. The breadth of information that can be captured is sufficient for everything we've learned about the project to date. On previous projects, much of the intellectual property was locked in the heads of the people on the project. The information in this form was hard to come by, and nearly impossible to understand. By capturing everything on "paper", we've all gained a better understanding of the problem. By having something to point at, wave your arms around, and mark-up, the discussions that used to take hours to gain consensus, now take minutes.

At the start, I was seriously concerned that the amount of paperwork would be overwhelming. That I'd be spending the rest of my career drawing cartoons in Visio or some other tool. SEEM has streamlined the process to the point where the documentation becomes an effective means of communication. We're still getting used to having

[5] The "oneness" of an object, high cohesion yields good architecture.
[6] The reliance of one object on another to accomplish a specific task, low cohesion yields good architecture.

everything captured, and it takes a minute or two to find information, but having it captured allows us to move forward at an accelerated pace.

Day 26

I forgot to mention yesterday that we've started to gain an appreciation for the continuity and traceability provided by SEEM. Since we wrapped up our first approximation of the user stories, we've been using them to drive the rest of the development process. Even our project plan is based on the user stories, as they provide an excellent basis for planning, estimating and tracking progress.

For each user story, we've identified the entities (nouns) in the problem domain and mapped them into sequence diagrams that accomplish the work of the user story. Then we examine the sequence diagram, and extract the roles and responsibilities for various entities into ERC Cards[7]. Any static relationships that we find between entities we also capture on entity-relationship diagrams. By the time we're done with all the user stories, all the entity's roles and responsibilities are flushed out and ready for the next step. Of course, it's not as linear as it sounds, and we keep going back to change our previous work, but I'll write more about that later.

Day 32

We've been reviewing the artifacts with the stakeholders, and they have the nerve to suggest we missed a few of their finer points. I feel comfortable joking about this as changes are really not a problem. In fact, by reflecting some of our learning back at them, we helped change their concept of the problem, and the eventual solution.

One of the interesting aspects of software development is that every project you undertake has never been done before: Everything is new. The consequence is that you don't know what you don't know! It sounds odd, but it's true. The only way to figure out a right way to accomplish a goal is to do it. Then decide if it was the best way or not, and re-do it if necessary.

I remember hearing in college that at least one expert recommended that you throw the first three versions away. While this is silly in practice, I understand the sentiment. At each step of the process, we realize where we went wrong only after we were done. Fortunately, SEEM's LPC[8] Loop takes this into account and actually leverages this fact of human nature to drive iteration. Everything we learned by doing, we captured in the journal either as a learning, or as an action item to go back and fix things. In some cases, we decided to move forward with a sub-optimal solution. After all, what we have works - it just isn't ideal. We captured that fact in the journal and will fix it when we revisit for the next iteration. In other cases, we decided to address the situation immediately, so we exercised the SEEM Systemic Iteration loop. We'll be trying that out tomorrow.

[7] Entity-Relationship-Collaboration Cards.

[8] Learning-Principle-Concept

Day 34

We’ve gone through our first exercise of systemic iteration. This was a very interesting process I’ve not used before. To start, we updated the artifact dependency graph we started on the second day Tom was here. He told us we’d need this! The dependency graph simply tracks the relationship between each of the artifacts created along the way. We had seen how to construct this graph from the initial set of artifacts created. We started with a product concept document provided by my boss, so that was on the far left of the graph. Each artifact we created thereafter was added to the right, and a line was drawn to represent the ancestry of each artifact. For example, since we used the product concept to generate user stories, there was a line from the product concept to the user story artifact. Similarly, we used our discussions with stakeholders for the user stories, so we linked the meeting minutes, or actual recordings of the conversations into the user stories. Later on, we had seen how the flow of events in the user stories drove the creation of the sequence diagrams, from which we had constructed the ERC cards. All of these showed up as dependencies in the graph. As more artifacts were created, this graph was be augmented. Tom talked to us about traversing these dependencies to manage change, but the elegance was lost on me at the time. Some of my colleagues, in fact, commented about wasting time to create a neat graph to illustrate dependencies that were obvious (I overheard someone remark that we should have taken degrees in graphic art rather than computer science). Now I understand the need for this graph, and appreciate the simplicity of the concept.

After we updated the dependency graph, we took a look at the changes we negotiated with the customer. Three of the users stories changed slightly, there was one user story added (discovered via the metaphor!) and the customer requested that we use a particular technology for our solution. For each change, we started with the dependency graph to assess the impact of the change. Starting with the artifact directly affected, we walked through the dependencies to determine if the other artifacts would be affected. For example, we had to add a new user story, which directly affects the user story artifact. Since the product proposal was used to create the user stories, we read though that to see if anything had to change. Since the new user story was consistent with the existing concept, no change was necessary. Going in the other direction, we know that the user stories were used to create sequence diagrams to assign the roles and responsibilities to objects in the system. Since we added a new user story, we had to add a new sequence diagram, changing that artifact.

Since we changed the sequence diagram, now we had to examine the dependencies to that artifact. Going back to the user stories, we knew we had already covered the change there, although this won’t always be true. In the other direction, we know that the new sequence diagram added a few responsibilities to the existing classes. Therefore, the ERC cards (which are dependant on the sequence diagrams) had to change. At this point in the development process, we don’t have anything past the ERC cards, so this was the end of the change effect. The other changes were handled in a similar manner.

Since the dependency graph portrayed the potential effects of the changes, and provided a means to manage the change, we were able to easily determine the effects of a change, and understand the consequence to the schedule and resources. Tom pointed out an interesting side effect of this graph. He told us that the number of perturbations induced by a change is a direct reflection of the quality of the system architecture. He explained that the dependency graph represents a perspective of the system architecture, and that changes that affect too many artifacts may be indicative of architecture problems. As he's been mentoring us through the process, this wasn't an issue for us.

Day 39

I had a project review with Sue, my boss, today. Naturally, she asked where all the code was, assuming we'd have a lot completed since we've been working as a team for so long. I laughed self-consciously, and explained that we just now understand the problem.

I showed her all the artifacts we created, and walked her through the problem analysis. She has seen many of the artifacts in different review sessions, but this was her first exposure to the completed analysis. Starting with the user stories, and ending with the ERC cards, we reviewed every aspect of the problem. As she is somewhat familiar with the problem space, she had no trouble understanding my presentation. She was astonished at the complexity of the problem that we modeled. Since we captured requirements from all the stakeholders, there were some interesting implications discovered. All the conflicting needs were reconciled and presented in our model. After just one hour of reviewing, she understood more about the problem we were solving than any of us had at the beginning.

Sue asked about hardware integration issues. I said that at this point we were still figuring out the problem, and didn't yet know how the hardware would figure in. I explained that hardware was a solution issue, and that would be a major part of the technology we considered. I also mentioned that we had spent considerable time with the hardware engineers gathering their requirements, so we felt comfortable that we understood their needs at a high level.

She apologized for her skepticism, and then asked how we were doing relative to the schedule. I explained that it was taking more effort than I expected to figure out the problem, but we were still slightly ahead of our original schedule (a schedule I thought was far too optimistic!) As I was leaving her office, she stopped me at the door and said she was pleased that she didn't have to fire me. She paused, then finished with a smile: "at least not yet."

Day 42

We've started mapping our problem analysis into the solution space. We spent some time reviewing the analysis with the hardware folks, and they had some interesting ideas and perspectives on the problem. Together we were able to consider hardware solutions that didn't make sense before we understood the problem. The hardware team gave us tons of input into the concept selection process, and made our jobs significantly easier. Using SEEM's concept selection process, we've been slowly working through the available technologies and selecting the right ones for our project. I've seen the Pugh process used in other contexts, but this is the first application I've seen for software. Basically, we list the various features we think are important along the left side of a matrix. Across the top columns, we list the different technologies we think apply. We select one technology as a reference, and then rank the relative strengths of the other choices to the reference. Using plusses and minuses, it quickly becomes obvious which technology makes the most sense. In fact, in many cases we didn't even have to finish the matrix as the very exercise of creating the matrix made the choice obvious. In other cases, we're still debating the finer points of the rankings. I suspect I may need to dictate a choice or two just to move forward.

Tom tells us that concept selection can be used anyplace there is a conflict or debate about a concept. Thinking back to our metaphor discovery meeting, I can now see how this might have applied.

To make sure we don't lose our decision making process, we've added the concept selection matrices to our repository, and to the dependency graph. That way, if we need to change anything, we'll be able to tell if any of technology choices need to be reconsidered.

Day 44

We reviewed the set of user stories today and selected the ones we want to implement in our first iteration. Most of our selections represent the essential behavior for the first release, but we included one that we considered high risk as we're not sure how we're going to do it. Hopefully we'll figure it out when we get there.

Based on the selected user stories, their relative complexity and risk, we put together a final estimate for the first release date. We also have a better feel for the technologies we'll be applying, so we submitted a hiring requisition for the other team members. Interestingly, we requested junior-level people to supplement the team. Since the SEEM process spells out what needs to be done to a reasonable level of detail, we don't need senior staff for the implement. We've also discussed the fact that junior-level people are typically more open to different ways of doing things, and we think they'll integrate better with the team.

Day 47

We got stuck today trying to figure out how to map our problem analysis into the technologies we've selected. We know what technologies to apply, but we're not sure how they fit into what we've done so far. We met with Tom to discuss the issue, and he reminded us of the need for traceability. He explained that the solution design phase includes all the same information as the problem analysis, but adds more detail, specifically around how to accomplish the responsibilities of the entities.

We walked through one example where we copied the sequence diagrams from the analysis into the design directory. Then we chose one diagram from the set, and started adding solution-domain entities. The flow of information is exactly the same, and even the problem analysis entities remained (although now they are subsystems or classes!) We added classes to realize the technology portion of the solution: things like linked lists, persistent records, and serial interfaces.

To clarify the linkage between the problem analysis and the solution, Tom suggested we color code the sequence diagram to illustrate the connection. Reverting back to the analysis diagram, we selected a transaction from the sequence diagram and colored it green. Returning to the design diagram, we colored it green to match. By comparing the two, we could easily see where the design classes were introduced, but we could also see where the sequence of events came from in the problem domain. The clarity of traceability was remarkable and really highlighted the value of understanding the problem first.

At this point, I started to understand the process from a higher perspective. It's really the same process repeated at different levels of abstraction. Even the diagrams are the same; it's just the level of detail that varies. It reminded me of a recursive descent language parser I wrote for a compiler class in college.

Day 50

We're almost ready to write code, but we were reminded to consider the Engineering Testing before the actual implementation. Following XP philosophy, we created a test plan for each of the implementation classes based on the CRC cards. This was my first exposure to writing the test first, and I approached the problem with some trepidation. After all, how can I write a test for something I haven't written? As it turns out, writing the test plan first helps formulate the boundary and error conditions you might encounter, and ensures you write code correctly the first time. I've never considered a test plan as a development tool, but it certainly works to your benefit if you write them first.

Day 51

We finally wrote some code today! We set up the compilers and the environment a while ago, and we've prototyped a few things just to be sure the technology was working as expected. But today we actually completed the first user story!

We started with the CRC[9] cards created from the sequence diagrams, and used them to outline the header file for the C++ classes. Then we filled in the bodies of the methods based on descriptions in the CRC cards and the sequence diagrams. We each took a set of cards, identified the required methods for our user story, and went to work. Since all the details were already thought through, writing the code was a no-brainer. Similarly, since the interfaces and interactions were all defined, when we brought the code together, it just worked. We even ran our test cases, and all but a few boundary conditions passed. No surprises, no muss, no fuss. We left work early, and I took the team out for beers. Sue bought, but I'll let her know later.

Day 57

We spent some time today automating the testing process. Since we're integrating continuously, and we're practicing collective ownership, it's become increasingly important to run our test suites regularly. Fortunately, since we're an embedded system without a user interface, automated testing is much easier.

We created a script-based test infrastructure that sends messages into our application, and examines the results. We were then able to write scripts to realize our test cases fairly quickly. Now, just prior to checking our code into the repository, we can run the test scripts against the code we've modified to ensure it's correct. The test scripts require

[9] Class-Responsibility-Collaboration

extra effort to maintain, but we're so far ahead of where I thought we'd be, it won't affect our schedule.

Day 63

We started integration with hardware today. I was expecting a nightmare of misunderstandings and miscommunication to result in incompatible hardware and software. To my surprise, everything integrated and worked with only a few minor timing glitches. The sequence diagrams that detailed the interface between hardware and software were invaluable in communicating the intent to both hardware and software engineers. The resulting hardware matched the specs perfectly, and the tests based on the sequence diagrams ran without a problem. We were pleased to learn that the hardware folks understood the sequence diagrams without any problem, and they were thrilled to have a detailed interface they could design to as well. They've been hinting that they're considering SEEM concepts for some of their processes as well. We recalled from the first presentation that SEEM is compatible with DFSS[10] philosophy for hardware and manufacturing, so we gave them Tom's business card.

Day 66

The first set of user stories is nearly complete. There are some tests yet to run, but the essence of the first iteration is ready. In hindsight, this is the easiest development effort I've ever participated in. Because we thought through everything up front, the coding process, and the integration were simple.

In some ways, I expected the effort to get more difficult as time progressed due to the interaction between the components of the system. For the first time, we're practicing collective ownership, something introduced by XP and promoted by SEEM. Collective ownership says that anyone can modify anything as needed. The consequence is that the code evolves much faster, and the quality is vastly improved due to the many eyes reviewing it. Also, since everyone knows the system well, they don't reinvent code when it's already been developed and tested. Since we have test cases for all the code, making changes is safe as you can immediately test the results of your changes to ensure you didn't break anything. Of course, this doesn't mean that its open season on the artifacts, its just that when you notice things aren't consistent, you can change them. Most changes need to be handled via systemic iteration, but the little things can just be fixed. For example, I was reviewing some code that Sandy, one of the new hires wrote. There was a section of code that I'm sure made sense to her, but it took me awhile to figure out. I made some simple changes to clarify the code. Fortunately, it was a change that was confined to the algorithm that implemented a particular method of a class. As such, it did not change the architecture. Exercising the systemic iteration process, I realized that there was no need to go and change any other artifact. I went ahead and added a few comments explaining the algorithm hoping the next person would have an easier time.

The other thing we're practicing is constant integration. Since the repository always contains working code, it's much easier to test new portions of the system. We still have

[10] Design for Six Sigma

to coordinate user stories to ensure the system evolves together, but that hasn't been a problem yet.

The cool part about constant integration is that we've been able to demo the system as needed without any prior notice. Sue brought her boss down this morning. Apparently she was bragging about our incredible progress, and her boss wanted to see for herself. They walked into my office unannounced to see a demo. Fortunately, my portion of the system was working at the time, so I was able to demo the application completed thus far. Her boss was quite pleased with the results, and Sue gave me a "thumbs-up" on her way out.

Day 75

Well, we demo'ed the program to the primary customer today. We completed and tested the essential functionality, and put together a formal release. Then we ran the customer acceptance test based on the selected user stories. Everything checked out fine, so we took it to the customer for their comments. We were especially pleased with our results as we delivered the first release 2 months ahead of schedule, and squeezed in 2 extra user stories.

Naturally, they wanted it right away, so we left them a copy with the preliminary user documentation created from the user story descriptions. I was pleased that upper management attended the demo, and was able to see our presentation, and the customer's reactions.

Afterwards, Sue and her boss dragged me aside and congratulated my team and me on our overwhelming success. This was the first time our company had delivered anything on time, and the fact that we were early with extra features was totally unexpected. Sue's boss had already heard from the sales people who attended, and they were ready to line up more customers based on what they saw. Everyone was very pleased with the results, and considered this a turning point for the company.

Sue had an extra twinkle in her eyes when she handed me a stack of envelopes, each with a team member's name noted on the front. I flipped though the stack to mine, and slit open the envelope while she watched. Mine held a bonus check, as I'm sure did the others. Glancing at the numbers, I realized I could now afford that new car I've been eyeing.

Day 80

Sue called me into her office today to debrief the project. I explained that the team was hard at work on the next iteration, adding in the next set of user stories for the customer. She asked what it would take to roll out SEEM to the rest of the company. Based on our success, they'd like to get everyone up to speed so they could be as successful as we are. We talked awhile about how to get everyone up to speed, and how we could spread the new expertise across the company. We settled on a starting point for the roll out, and I headed out the door. As I reached for the knob, she said, "One more thing." I turned around, and she slid a new nametag for my cubicle across the desktop. My name was across the top, the same as my current nametag. But on the bottom was a new title, Director of Software Development.

References

1. The Standish Group International, 2000. The Standish Group, CHAOS Chronicles
2. Kruchten, Philippe. 1999. *The Rational Unified Process*. Addison-Wesley. ISBN 0-201-60459-0
3. Beck, Kent. 2000. *eXtreme Programming eXplained.* Addison-Wesley. ISBN 0-201-61641-6
4. Schneider, Winters. 1998. *Applying Use Cases: A Practical Guide*. Addison-Wesley. ISBN 0-2013-0981-5, See chapter 8 for further information.
5. Booch, Grady. 1994. *Object Oriented Analysis and Design with Applications.* 2nd Edition. Addison-Wesley. ISBN 0-8053-5340-2. See discussion starting on page 12
6. Cockburn, Alistair. 2002. *Agile Software Development.* Addison-Wesley. ISBN 0-201-69969-9
7. Fowler, Scott. 2003. *UML Distilled Third Edition – A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley.
8. Bredemeyer, Dana. See http://www.bredemeyer.com.
9. http://www.ArchSynergy.com