In-the-SPIN

Newsletter of the Boston SPIN

Issue 56 Spring 2006 **Editor: Judi Brodman**

From the Editor

By: Judi Brodman, Editor

Well, here we are in the heat of summer with our last our last SPIN year far behind us! Hopefully, this issue will provide plenty of late summer reading.

Let's start with the introduction of our new Steering Committee members. Remember that they are responsible for everything concerning SPIN – no small task!

In the Contributor's Spotlight this issue, there is an article by Norm Daoust, ***The Unified Modeling Language (UML): What's in it for you?*** Norm is an expert on this subject and shares some of his knowledge with us through this article. Please continue to submit articles for publication – this is a great place to have your views and material read.

We have a new section in this issue – Book Reviews. The two books reviewed, both recently published, address the issues of estimating and planning as well as sustaining Agile development. So take a look at what our reviewers thought of these new books. If you would like to be a reviewer, please let us know!

Also included in this issue are the results of some of the Roundtables (RT) starting in January 2006. The RTs continue to be very popular and the sharing of experiences and expertise continues. If you haven't attended one or haven't been a facilitator, you really should try it.

If you have any comments or questions about our *In-the-SPIN* newsletter, please write me at:

brodman@logos-intl.com

Boston SPIN
Software Process Improvement Network
Established January 1993

IN THIS ISSUE . . .

From the Chair

By: Sean Gaines, Chairperson

If it's not broken, don't fix it.

Working on older cars I learned the true wisdom of this statement. Break enough rusted bolts off in your zeal to do standard maintenance and you learn the lesson first hand. But in most of my activities, and especially with regards to Process Improvement, I could never accept this philosophy. Just because "it works" must the process be cast in stone? Just because a process does not appear broken, must we be complacent and accept that it is the best it can be? Are we afraid to challenge the status quo? My thoughts are that all too often, it is not good ideas that are in short supply; it is the courage to voice new ideas.

Hear the voices of courageous leaders.

Boston-SPIN provides the network to meet peers in this challenging field who are interested in sharing their experiences. It is an opportunity to hear how others found the courage to voice their ideas and effect change – something I find exciting. It is the chance to mingle in a group of ambitious like-minded professionals – something I look forward to every month. Being inspired to help with the efforts required to keep Boston-SPIN a thriving group, I now find myself contributing as the Steering Committee Chairperson. Which reminds me of another famous phrase: be careful what you ask for!

Be courageous and volunteer your voice.

For those of you who attend the September meeting, the first thing you will hear me voice are the many thanks that should go out to all of the volunteers who make it possible for the rest of us to enjoy another great year. We are expecting new challenges this year as we expand our host sites, but this group is so incredibly well organized that I am confident we will rise to the occasion as needed. However, if the time has come where you are ready to rise up and volunteer to help with any of the many tasks – please be sure I hear your voice! In addition to our speakers, we need a constant stream of courageous contributors to keep Boston-SPIN a thriving group, so if you can, please volunteer your assistance.

A Special 'Thank You'

Let's give a special round of applause to Sheila Lynch who has been our MITRE SPIN coordinator since September 2002 and completed her task this last June. Sheila was the one responsible for making

sure that everything was set up for the meeting including all the audiovisual needs of our speakers, Roundtable needs, even MITRE guards. She was at the meeting facility long before we arrived and left long after we had all gone home!

Thanks Sheila for a job very well done!!

Also, thanks to our 2005-2006 Steering Committee and all the special committees for a GREAT year!!

2006-2007 Steering Committee Members

In June we elected a new Steering committee. Some have been members on the Steering Committee before – some have not. Please support them by attending meetings and by volunteering for some of the activities that help this organization run well!

The Steering Committee contact information can be found on the Boston SPIN web page: http://www.boston-spin.org/

Chair: Sean F. Gaines

- President of the UMASS Stereo Co-op and the UMASS Hang Gliding Club.
- 10+ years of Small Business experience as Owner and Master Technician of Underground Sound Inc., an audio/video repair shop started while in college.
- Board of Directors of the Amherst Chamber of Commerce.
- Field Engineer providing on-site hardware/software support of GE's CALMA CADD systems including (early) jumbo electrostatic printers and plotters.
- QA Manager of in-house and offshore QA resources for a software development company.
- Customer Support Manager and Senior QA Process Engineer for a mechanical engineering firm.
- Database Design consultant for Harvard's Office of the Governing Boards.
- Chief IT Manager + DB System Architect for a company specializing in the electronics industry's need for EOL, LTB, excess and scrap Inventory Management.
- Current software process improvement interest is Business Process Engineering - converting business needs into achievable technical specifications

Vice-Chair: Maxine Crowther

- Currently Sr. SW Quality Engineer with iRobot Government and Industrial Division in Burlington, MA – a CMM level 3 company. They are looking at AS9100 (AS9006 for SW) and CMMI certification in the next year.
- Past SPIN Board Member 1997-1999.
- Member ASQ since 1995.
- MS in Quality Systems from the National Graduate School.
- Special interest is in SW Process, SW Metrics

Secretary: Carol Perlitz

- completing second term as SPIN secretary
- Sr Mnr of QA at Juniper Networks
- most recent presentation at ASQ Boston software section was on planning testing based on internal and external factors.

Treasurer: Barry Mirrer

- more than 20 years of software quality experience and is currently Director of Quality at Outcome, which designs and implements web-based medical studies and registries
- past SPIN Program Chair (2001-2002), has been a Boston SPIN member for about 10 years, has taken finance courses at Harvard
- published author and award-winning speaker

At Large: Steve Isenberg

- currently works for Nortel Networks as a Sr. SW Quality Engineer
- is a current board member, serving as treasurer, of the Greater Boston Network Users Group. He has 18 years of volunteer involvement with this group, including having served as vice-president

At Large: Ron Kay

- completing second term as At-Large member of SPIN steering committee
- served 4 terms as SPIN treasurer
- currently works at Intel in software QA

Program Chair: Michelle Gross

Michelle's day job is Quality Assurance Manager for the speech-recognition product, Dragon NaturallySpeaking, at Nuance Communications, Inc.

Michelle used Dragon NaturallySpeaking to dictate this brief bio.

Contributor Spotlight

The Unified Modeling Language (UML): What's in it for you?

Norman Daoust, founder of Daoust Associates, an information modeling and systems integration consulting firm

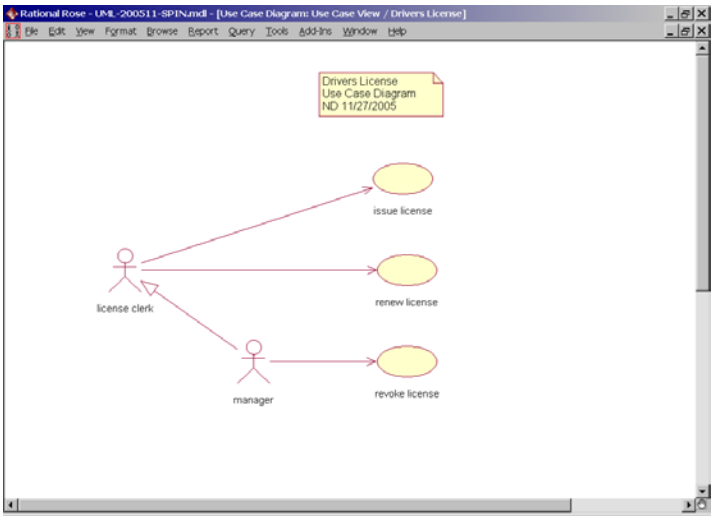
Introduction:

The Unified Modeling Language (UML) was published first in 1997. Its use in software development projects has quickly spread throughout the world.

In this article, you'll see an example of a use case diagram and a class diagram and a brief description of some of the other eleven diagrams in UML Version 2.0. Finally you'll learn why you should consider using UML.

Use Case Model:

The use case diagram is frequently used to capture functional requirements for a system and to delineate the scope of the system. Here's an abbreviated example.

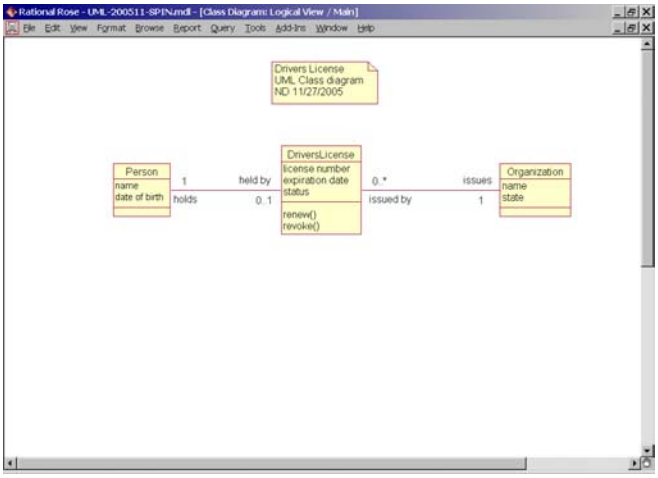


The diagram indicates that a license clerk (an actor) can issue and renew driver's licenses (use cases). A manager can perform both of those functions as well (the line with the open arrow from a manager to a license clerk indicates the manager is also an actor in all the use cases of a license clerk, but can perform addition use cases as well) but can also revoke a license.

The real power of use case modeling lies in the text portion of a use case model, but unfortunately the diagrams have become the focus of most of the publicity surrounding use case modeling.

Class Diagram:

The class diagram is one of the most widely used UML diagrams. Here's an abbreviated example.



The rectangles represent classes. The class name (e.g. Driver's License) appears at the top, followed by its attributes (e.g. license number), followed by its operations (e.g. renew). The lines connecting two classes are known as associations. To read an association, you would say "An Organization issues zero to many Driver's Licenses. A Driver's License is issued by one Organization."

Remember a diagram does not a model make! The text behind the diagram is sometimes the most important part of the model. For example, the text definition of the status attribute of a drivers license might have as a definition: "The current state of a person's drivers license: active, suspended, revoked, or expired."

Other UML Diagrams:

An Activity Diagram is an all-purpose diagram good for modeling workflow and time sequences of interactions. They're frequently the diagrammatic glue that can be used to bind together many different views of a system into a single diagram. Object diagrams are similar to class diagrams, however they include sample values for attributes; this is helpful to illustrate concrete examples. Statechart diagrams are used to illustrate life cycle states. Imagine the possible lifecycle of a drivers license: issued into an active state, being suspended and subsequently reinstated, and finally being revoked. Sequence diagrams are used to illustrate the passing of messages back and forth between objects. Deployment diagrams are used to illustrate the different hardware devices in a system, and optionally the software components installed on each one. Packages are a mechanism to group together related items of a kind. For example, the classes of a class model are often partitioned into several different packages.

Summary:
The Unified Modeling Language has become a powerful tool for communication in the software development process. If you believe the adage "a picture is worth a thousand words", it's obvious why UML diagrams have become so popular. If you couple high quality text behind the diagrams with high quality UML diagrams, you have an excellent combination to for communication in software development projects.

Why should you use UML?

It's a tool to improve comprehension and communication. Readers find it easy to review, critique, and detect errors and omissions in diagrams. If you believe you discovered items left out of the two diagrams in this article, you've proved my point!

It can assist in standardizing the software development process of an organization.

The same notation can be used to model from different perspectives: the perspective of customer of the system; the perspective of the software system itself.

The notation is a standard widely adopted throughout the world.

How to Get Started:

Many organizations take a "wade in the water" approach. They use two or three of the UML diagrams in a small project. They then spread the use of those diagrams throughout other projects and in their next project use a few more types

of UML diagrams. Others take a "dive in head first" approach. If you opt for this approach, I strongly recommend you obtain high quality training first and then mentoring support throughout the project.

Conclusion:

This article has given you a very brief introduction to the Unified Modeling Language. I encourage you to explore it a bit more and begin to use in your next project.

References:

www.uml.org. The web site for UML from the Object Management Group. The specification is available from this site. The site also includes a list of UML modeling tools.

UML Distilled, 3rd edition, Martin Fowler, Addison Wesley, 2004, ISBN 0-321-19368-7. An excellent introduction to the Unified Modeling Language. It's a short paperback, easy to read, and covers the 20% of the UML people use 90% of the time.

Book Reviews

Bruce Taylor, the owner and principal of WorkingInUnison, an Organizational Development consulting firm located in Acton, Massachusetts, was very generous with his time and reviewed the following book for us!

Thanks, Bruce!!

Agile Estimating and Planning, Mike Cohn, Prentice Hall, *Robert C. Martin Series,* 2005

When I started to read Agile Estimating and Planning, I planned on writing a review that said, "Well, this is a good theoretical book, but there's not much guidance for the practitioner." Then I noticed that the book was filled with realistic examples of planning and managing agile projects. So, then, I was going to write a review along the lines of "It's all very well to give us a cookbook approach, but we need to understand why this works." And then, I noticed that every chapter explained exactly what managers need to achieve, why they need to achieve it, and how to think about management problems in an agile way.

So now I'll have to write, "This book is a remarkable blend of the theory behind agile management and extremely pragmatic advice about how to make it work; it was clearly written by a manger who has been through many agile projects and has reflected on what makes them successes or failures." I don't often have an opportunity to write a review like this one.

If you're a manager starting to learn agile development, one of the scary barriers is that none of the techniques that you learned in classical waterfall projects apply any more. You no longer have the luxury of a months-long planning process up front; you can't measure progress by date-driven milestones; and you can't predict progress by estimating lines of code per person per day (not that this ever worked very well.) As the man says, "If this doesn't terrify you, you just don't understand the situation."

Rather than review this book chapter by chapter, let me introduce the three most important concepts in the book - the essence of agile planning, as it were:

- Estimating Size with Story Points or Ideal Days

  When you need to decide how big a task is, you can choose either of two methods. First, you can look at the story cards (you did write story cards, didn't you?), and assess the difficulty of each card in "points." Given the number of points assigned to the cards and the skill of the staff available, you can convert to expected effort. Or, you can rely on the fact that engineers are usually pretty good at estimating their work capacity on "ideal days," when no one interrupts them and nothing goes wrong. So you can ask for an "ideal day" estimate and multiply by a reality factor that you will have to determine empirically to determine the amount of effort required. Each of these techniques is appropriate in certain circumstances, and Cohn helps you decide which technique is more likely to produce an accurate answer.

- Coping with Feature Overload by Customer Value

  There are always more features required than you have time and energy to implement: this is a basic law of software physics; and the classical way of dealing with this in the waterfall model is to flog the programmers harder to squeeze the features into the milestones that were laid out six months ago. Since an agile development cycle is two to six weeks long, the problem is different: how to choose from among competing features the ones that will go into the next iteration. Cohn argues that this should be decided (mostly) by the feature's value to the customer, and that value can only be determined by the customer team. For features that are too big to fit into one iteration, he gives some heuristics for decomposing it into logically separable pieces that build on each other. I might quibble that some feature choices are affected by architectural requirements, but Cohn is essentially right in using customer value as the touchstone.

- Team Velocity

  I don't know if Cohn invented the term "velocity," but I like it and find it very expressive. Roughly, it means, "How much work can this particular team complete over this particular stretch of time?" If a manager had a reliable, stable velocity number for each team, then scheduling could be reduced to estimating effort; but Cohn shows that velocity is an imprecise number at best and varies over time and with circumstances beyond the team's control; so the best a manager can do is to estimate the probable range of the velocity and use the high and low ends of the range for planning. Estimating velocity is clearly a skill that good managers have to learn over time, but Cohn gives some very good

rules of thumb, and ways of checking your estimate against reality.

I need to make some observations about Agile Estimating and Planning as a book. Cohn writes very lucidly and, he clearly understands his reader very well - nowhere does he overwhelm us or condescend to us. His editor needs to stand up and take a bow: the book is clearly and logically organized and I could find any topic in a few seconds by searching through the table of contents. The index is organized by exactly the topics I would want to find and complements the Table of Contents very well.

So all in all, this is a timely book written in a very straightforward way by an author who clearly has lived through many, many agile projects and has gained insight and experience that he is willing to share with us. I wish that I could find at least something to pick on, but I honestly can't. If you're a manager facing the scary transition to agile development, you really, really need this book on your desk.

Agile Estimating and Planning by Mike Cohn

George J. Schlitz, PMP, CMS, was also very generous with his time and has given us a second perspective on this book!

Thanks, George!!

"What is your estimate for analysis, development, and testing of these enhancements?" No question filled me with more feeling of exposure and dread in my early days as a developer. Repeated many times, the answers would result in project plans that were held over our teams' heads for months. Uneasiness about the plan started immediately, and only increased as reality unfolded. This became my main motivation for becoming a project manager - there must be a better way to estimate.

"Planning is everything. Plans are nothing." These words of Field Marshal Helmuth Graf von Moltke introduce the first chapter, and their importance and relevance to software projects is clear throughout. Mike Cohn provides us with a well organized, smooth-flowing read that simplifies the mysteries of project planning. As the title declares, the book was written for agile projects, but I believe that the lessons and examples inside can provide a pragmatic, repeatable framework for planning for changing project environments and debunk the common misconceptions that agile projects are short on planning (in fact the opposite is true).

Sections of the book include:

- An introduction to why planning is important, what constitutes a good plan, what the problems with planning are, what agile planning goals are, and why traditional planning and estimation often lead to unsatisfactory results
- Estimation techniques that focus on estimating size and deriving duration, group estimates, and how and when to re-estimate
- Planning with a focus on value, which includes guidance on prioritization using various common financial valuation methods, modeling financial return, assessing and prioritizing desirability, and splitting features into more manageable sizes
- Scheduling and planning iterations, assessing team progress (or Velocity), dealing with uncertainty, and planning multiple-team projects
- Monitoring the release and iteration plans, and communicating the plan and progress
- An explanation of the reasons why the recommended approach works.

Managers of agile projects will gain an effective toolkit for planning and estimating, as will managers who need to gain control of projects in complex/changing environments where traditional methods haven't produced satisfactory results. I believe this is a must-read – it has found a place near the top of my often-referenced books.

Anyone who believes, as I did, that there must be a "better way" to estimate and plan projects than the complex and unreliable methods that we are used to, and anyone who is interested in learning about agile methods will gain from a planning perspective - and maybe will gain new respect for well-run agile projects.

I did not submit this review until I gave most of the techniques Mike recommends a try on actual projects - the effects on the entire project team (including stakeholders) were quickly visible. The estimation techniques provided a common language for the cross-functional teams, and the scheduling simplified the process of release planning and prioritization. Each month, as described in the book, estimation and planning became more accurate and repeatable.

Sustainable Software Development: An Agile Perspective, by Kevin Tate, Addison-Wesley, 2006

Bruce Taylor, the owner and principal of WorkingInUnison, an Organizational Development consulting firm located in Acton, Massachusetts, was very generous with his time and reviewed the following book for us!

Thanks, Bruce!!

The entire field of computer programming is only about fifty years old, the notion of *organized* programming dates back only twenty or thirty years, and those years have been crammed with a succession of software development methodologies, each of which would definitively fix "the software productivity problem." Some worked a little, some worked poorly, and some were disasters, but they all had one thing in common: they relied on making programmers work harder and harder to achieve whatever gains the methodologies promised. They were all the equivalent of telling the drummer on the Roman galley to pick up the pace a little.

But now comes a sea change: one of the cornerstones of agile development methodologies is that programmers are humans, have a life outside the cubicle, and can't work sixty-hour weeks for months on end without breaking down. Agile tells programmers and managers alike: "This is a marathon, not a sprint: you'll accomplish more if you set a pace that you can keep up for year after year after year." And the programmers and managers reply, "That's a great philosophy, and we like to hear it, but specifically what should we do to implement it?" And here, most books on agile development fall silent.

One Exception is *Sustainable Software Development: An Agile Perspective*, which describes the functioning of a software development organization that can achieve high productivity for very long periods of time, while energizing the programmers. Mr. Tate starts by reviewing the current state of the practice, which is unsustainable because it tends to burn programmers out before the products ships. He reviews all the factors that can contribute to the project from hell: poor leadership, unstable requirements, late integration, and so on. None of this will come as a surprise to anyone with experience in the field, but it's nice to see it summarized.

Now he shifts to the principles of sustainable development, and I was ready to read the usual set of vague platitudes. But the title of his first section made me take notice: "Why Principles are More Important Than Practices." The argument of this chapter (which he carries through the rest of the book) is that the recipes for agile development are all well and good, but if you don't understand what you're doing and why you're doing it, you're not likely to do it well. Hooray! I think that's the first time I've heard that sentiment expressed so succinctly - at least in print. The principles he lists are common to most agile developments: maintaining a working product, continuous integration, the importance of refactoring, but he casts them in the context of sustainability, and that makes all the difference.

And finally, he follows with the practices themselves, and he is remarkably detailed in his treatment. He lists the usual practices of user teams, pair programming, and the rest, but he also talks about the importance of choosing the right tools, and of good configuration management, and even of coding, documentation, and design standards. Because Mr. Tate puts the practices in the context of the principles, it's always clear why a practice is important, how you would adapt it to your own organization, and when you might skip it altogether. Indeed, this section could stand alone as a "how to" book on agile implementation, better than most books on the market.

They tell me that, in any review, you're supposed to find three things that could have done better. So, keeping in mind that I think this is a great book, here are my suggestions for improvement. First, the book doesn't really address the macro-level causes of unsustainable software development: the relentless, unreasonable pressure to be first to market with a product, no matter how flawed. It's true that programmers and managers can't influence that pressure, but the book would be better if Mr. Tate at least explained the economics that drive the industry to implement current bad practices.

Second, his definition of "culture" is too limited to be useful. I hesitate to fault the book, because a description of organizational culture is a rather large book in itself, but it would have been nice to see a discussion of degree of risk taking, or traditions of transparent decisions, or any of the other cultural factors that make agile development possible.

And finally, the book tends to minimize the difficulty of making the change to a real agile environment. As many of us have learned, it takes months to introduce agile techniques into an organization, but it takes years to change the organization itself, and Mr. Tate's treatment of change makes it sound entirely too easy.

There, I've made my three suggestions for improvement, and now I'll remind the reader that they're relatively minor criticisms of a very good book.

The final appealing attribute of the book is that it speaks to the whole organization. Programmers will find material that helps them change their design and coding skills; managers will find new ways of planning, organizing, and allocating work; and executives will find strong arguments for adopting new, more effective, more sustainable software development policies, even in the face of the pressure to produce more and more with less and less.

Roundtable Synopsis

January 17, 2006

Negotiation skills for SW developers

Facilitator: Moshe Cohen, The Negotiating Table

Well-intentioned people, working together to accomplish common goals on projects often find themselves at odds with each other over the means to accomplishing their objectives. Understanding and resolving the differences between them is critical to the successful completion of the project.

For example, a software engineer and SQA person might both have the success of the project in mind but might differ substantially on the steps required to achieve that success. In an effort to deliver a quality project on time, especially on a troubled project, the engineer might check out a huge number of modules, essentially bypassing the version control system, and might occasionally correct defective algorithms quickly without changing the corresponding documentation and without getting approval for changes to the spec. SQA, looking at the same project and also wanting to deliver a quality project on time, might react with horror to such practices, knowing how much trouble they can lead to on the project over time.

In resolving the differences between them, SQA and the software engineer must first realize that there are a few right an-

swers in negotiation. Each person is approaching the situation from a different perspective and is trying to satisfy different interests along the way. Rather than take positional views, such as "change the code back" or "leave me alone," each must endeavor to try to understand the interests of the other. This is not a win-lose game. If they fail to resolve their differences, the project might fail to meet the customer's objectives and they might both lose out.

Any discussion in a negotiated process starts by considering each party's alternatives if they can not reach an agreement. So long as either party feels that they can achieve acceptable alternatives for themselves without negotiating with the other, they will have little motivation to negotiate. On the other hand, as they discuss their differences, find out each other's interests, and invent options for resolution that are better than their no-negotiation alternatives, the likelihood of a productive resolution of the matter increases. To help in discussing and resolving the issues, the parties can also benefit by looking together at objective criteria, such as precedents from other projects, customer communications, and industry standards.

Key to the success of any negotiation effort between SQA and the engineer is their ability to communicate with each other. They need to develop skills to express their interests, draw out the interests of the other party, and most importantly, listen to the other person. Negotiators often get so fixated on expressing their views to the other party that they overlook the fact that the best negotiators are generally the best listeners. In working out their differences through an interest-based process, not only might the engineer and SQA complete the project successfully, but they might also build a different and more successful working relationship on this project and in the future.

Function Point Analysis: Myths and Realities

Facilitator: Michael Bragen, CFPS, Software Productivity Research, LLC

Function Point (FP) Analysis is a methodology for quantifying software in terms of the "Function Point" metric. Through a process of functional decomposition, the FP analyst uses a set of rules established and maintained by the International Function Point Users Group (www.ifpug.org) to attach weighted values to the logical transactions and data groups that make up an application – or a set of requirements for an application. FPs are always derived from the logical (i.e., user's or specifier's) point of view, and as such are consistent regardless of the technology or development methodology used to build the system. The FP size of an application (or development or enhancement project) is useful in many types of software engineering management activities: estimation/costing, comparative productivity or quality analysis, defect density prediction, etc. CMMI and other process management disciplines recognize the function point metric as a valid and consistent approach to sizing.

Questions from participants addressed aspects of the counting technique (a relatively easy-to-learn set of rules and concepts), applicability (regardless of technology, FPA can be applied to any kind of software) and consistency/reliability (the approach is not perfect, but a high degree of reliability in measurement can be achieved with certified function point specialists – CFPS).

Another issue that was raised is how the FP metric is used (or misused) in organizations. The following fictional exchange illustrates an important point about what function points are…and are not:

Ask the CFPS (Certified FP Specialist)

Why is it that when I use FP methodology to size an application, the number comes out seemingly high, sometimes?

We had a system developed for $120,000, based on the client's budget. But, when I did the FP count, I came up with a size of 195. Our CFO thought that was much too high for such a small system.

I looked at my count again with the project manager knowledgeable about the project, and he thought he would have come up with the same count (at least within 20% of the number I got when counting.) So, I don't think I did the count wrong. Our CFO is asking why the counts come out higher than he expects. Not just for this system, but in general. He gave the example of counting an application like Microsoft Word, or some other prepackaged software that will have market value of $500, but the functionality count would obviously be much, much higher.

- Confused

Dear Confused –

I will attempt to clarify this with a simple analogy. First, consider that a "Function Point" count is a unit of measurement, no more, and no less. It was designed to be a so-called "pure metric" like "liters" or "inches" or "square feet." Let's take the latter example, a metric for area, and compare two rental properties measured in this way.

<u>Structure #1</u>

Area: 300 square feet

Location: Rodeo Drive, Beverly Hills, California

Construction media: brick, glass, steel, electrical service, piped in music, expensive carpeting

Architect: Frank Lloyd Wright

Usage/Purpose: Jewelry boutique

<u>Structure #2</u>

Area: 300 square feet

Location: State Road B5, Luck, Wisconsin

Construction media: pressure-treated lumber

Builder: Joe and his son, Bill

Usage/Purpose: Hay and fertilizer storage shack

Please note that both structures are measured (quite precisely!) to have the same area, in the chosen metric (square feet.) Consider the following comparative measures:

	Structure #1	Structure #2
Construction cost	$250,000	$2,500
Annual rental cost	$60,000	$1,200
Owner maintenance	$12,000	$200
Cost to build per sq. ft.	$833	$8
Cost to rent per sq. ft.	$200	$4
Rental profit/sq. ft.	$160	$3

Here are some questions and comments to address with your CFO:

- What makes him think that the function point size "should" be a certain amount?
- Unless one has a sense of what goes into a function point count, is it reasonable to consider "feelings" for what is "high" or "low"?
- Does he understand that function points are simply a quantification of required capability? They are NOT intended to be used as a simple factor for costing.
- IT IS NOT MEANINGFUL TO COMPARE RANDOMLY SELECTED PROJECTS IN TERMS OF AVERAGE COST PER FUNCTION POINT.

In view of the above example that would be equivalent to the statement that "Buildings cost an average of $420 per square foot to build." (!)

There are MANY factors (some illustrated in my example above) that influence the cost or value of producing or purchasing applications. It is ABSOLUTELY unreliable to apply function points as a measure of cost/value BY THEMSELVES, without considering all four of these key factors:

- Size (i.e., Function Points)
- Complexity (e.g., algorithmic, structural, data relationship complexity)
- Capability (e.g., environment, tools, media, risks, experience levels, etc.)
- Process (i.e., work process life cycle activities to produce the software)

It is very risky for any organization to attempt to misuse FPs in the way you suggest they are being used! FPs can be used as the base-scaling factor for estimates or application comparisons. There is no meaningful "average cost per function point" in industry; any more than there is an "average speed limit" on the highway. Of course that value can be computed from available data – but don't try to use it to fight a speeding ticket!

Exploring the technical aspects of automated software testing

Facilitator: Jean-Pierre Boissy

The roundtable began with a breakdown of two categories of common automation tools - one being GUI tools from companies like Mercury, Segue, and Rational; and the other, higher level languages like C, C++, Java, and C# for use with API testing. The discussion started with developing automated tests with the GUI tools and quickly highlighting the maintenance issues surrounding "capture-playback" as opposed to programming the scripts themselves.

One participant felt that the testers should simply recapture the tests instead of performing code maintenance. This issue, when considering large amounts of captured test files, appeared to lose the benefits of automated testing as the tester would be redoing the manual tests just to recapture the automated test files.

This topic further lead to a discussion about locating GUI components through x-y coordinates (capture playback) contrasted with identifying these components programmatically using object names/tags especially when considering that the screen sizes used by all customers would be variable.

Following these basics of GUI testing the roundtable addressed using higher level languages to test APIs. A question by one of the participants was whether API testing was like unit testing. This point lead to elaborating that both GUI and API testing should be done together and whereas unit testing focused on just a developers code module(s) the regression, performance, and stress testing QA runs exercises all of the modules working together. The API tests then could be used to test this interaction among modules developed separately.

The test tool "footprint" was then discussed - the footprint being defined as the resource consumption by the test tool itself. Installed GUI tools use CPU, memory, and bandwidth that will impact a system's overall performance. This suggested that monitoring and accounting for both system resource usage by the test tool and the application under test should be clearly separated when compiling test results. Further the question of valid testing was covered considering the use of one computer sharing resources, driving a test tool and the application under test, as opposed to many computers each running separate manual tests. This issue was considered and the cost and managing of many machines running manual tests seemed to slow down the testing process and did not seem to be feasible in a today's testing environment.

The design of automated tests was then covered and it was suggested that if a software defect was found manually it should be incorporated into the automated test suite. This implied an extensible test suite where each test would start in a known good state so differences could be logged and measured. Further exception handling would have to be built into the design to avert "hanging" the test suite's progress when it uncovered system or application defects.

The roundtable discussion ended with a reminder that developing test automation should be viewed as a software development project that requires its own specifications, source code control, and testing. Archiving post ship tests was also recommended so that they would be readily available to test maintenance releases.

April 18, 2006

Use Case Modeling: Requirements Panacea or Buzzword de Jour?

Facilitator: Norman Daoust, Daoust Associates

The group included people who had utilized use cases for several years, those with a small amount of experience, and those interested in learning about the topic.

The group reviewed a simple example use case diagram for motor vehicle registrations for a fictitious state motor vehicle department. The example included the text form of the use case as well as the diagram.

Use case diagrams are good for obtaining an overview of the functionality quickly. They're useful for managers and others looking for a high-level overview of the system. It's easy to spot missing actors and missing use cases. They're also used to partition the system, allowing those interested in a specific portion of the system to review just portion. Use case text is good for capturing the details.

The bulk of the discussion consisted of questions and answers.

How long have use cases been around?

Since the 1980s when invented by Ivar Jacobson in Sweden for a large telecommunications project, but they've become more widely used since the late 1990s after they were incorporated into the OMG's initial specification of the Unified Modeling Language (UML) in 1997.

To what level of detail should they be written?

It depends on the length of the project and the criticality of the system. Creating use cases can take from five minutes to two months.

Who drives the use case requirements?

At one organization it was the product manager acting as the voice of the customer.

Who write the use cases?

In one instance the developers and quality assurance engineers wrote them based on a loose requirements document.

Are UML diagrams the only way?

No, attendee has used strictly text based use cases.

What tools are used?

The tools for creating use case diagrams can be divided into three categories: (a) flip charts, white boards, yellow stickies; (b) Smart Draw, Visio, and other drawing tools; (c) UML modeling tools such as Enterprise Architect and IBM/Rational's products. The Object Management Group's UML web site contains a page listing UML modeling tools: www.uml.org/#Links-Tools.

How do you know when you're done?

When the stakeholders agree if a system was built with this functionality it will meet their needs, and the system architect agrees there's enough detail in the use cases with which to design the system.

The group at this Roundtable was interested in knowing about techniques for project estimation and development of better cost and schedule estimates.

Several estimation methods were offered: 1) using the knowledge and experience gained by seasoned practitioners from similar projects in the past, 2) knowing the requirements, 3) using metrics collected from past projects, 4) the Delphi approach using domain experts and experienced facilitators, and 5) using function points for the scope of work, which calls for training to be successful.

Familiar problems encountered in meeting cost or schedule, caused by the estimation process, or lack of it, were presented as were issues related to management, such as difficulty in convincing senior management of the validity of the estimates, management yielding to market pressure to provide estimates too early, being bound by a corporate mindset, or being held to estimates with no flexibility to change with the circumstances.

Other causes of estimation problems could be the data used, for example, using mean estimate data with no variability factored in, using scaling that is nonlinear, or having no historical data because of innovation, that is, attempting something new. Failure to perform risk analysis and failure to compensate for risks were other factors mentioned. A risk example would be vendor dependency – the vendor can't deliver as promised.

Misunderstanding dependencies of parts of the project, or changing, misunderstood, and unclear requirements could all have a negative impact on good estimation. Problems with estimation cause misallocation of resources, cost overruns, and missed milestones.

Participants offered ideas on what could be done to help overcome some of these shortcomings of estimation: 1) Negotiate with the customer to set expectations, 2) Pay attention to customer needs, 3) Revisit the initial estimate after more requirements analysis, 4) Emphasize communication between and among departments and teams, 5) Establish contingencies for risks that materialize, which may impact the cost and schedule, 6) Try phase estimates, e.g., quarters, instead of years, 7) Make the estimate work-product based to gain reliability and Earned Value, 8) Use methods such as a Mini RUP (Rational) or Business Process Modeling (BPM), where the modeling produces executables, and 9) Let stakeholders set priority and realize a Use Case from the output.

Feedback from the Roundtables showed that more in-depth information is desired concerning solutions to these estimation problems, more focus on one or two methods of estima-

tion, and the plusses and minuses of various project management tools, based on experience with them.

June 20, 2006

Measuring the Quality Software Level

Facilitator: Stephanie Beach, Interlocking Applications

There have been many articles and discussions on measuring the Quality level of your software. We've all read them. Right now, I work with the typical, "Test Coverage", "Defect Density", "Putnam-Raleigh Defect curve" and a few other measuring tools.

But when our customers use the software we are testing, what will they say about our product? Does their expression accurately reflect what our "Final QA Report" indicated? What IS the level of quality? How can we quantify this?

This discussion examined the different tools and measuring devices that we have available then tried to determine what combination of these is appropriate in measuring the quality level.

As we began, we quickly moved to the customer experience. We decided to take on the "Customer Viewpoint" approach and work backwards toward our metric. As you know, we can deliver all sorts of technical jargon but if the customer still feels that the software "isn't doing its job", then all the metrics in the world won't support your 'good quality' conclusion.

Therefore, we determined that an approach to take is to map out the customer's functional spec, their acceptance testing, and their Alpha-testing plan with the tests that our QA team ran over the last few test cycles. As the customer performed specific tasks, we would determine how closely our tests matched their exercises.

We start at a base line on a graph. Y-axis is Quality level: High at the top, Low at the Bottom. X-axis was grouped the categories discussed below.

Then for each activity that the customer performed that matched our test cases, we would move up 1 point.

If the area under test was a major deliverable, determined by the customer and product management, then it could go up 2 or 3 points.

For every test that failed, we drop 5 points (or more) and for every test we missed (one they performed but we did not) we could drop an additional 5 points.

This gave us a quick baseline as to how well our testing matched the customer's expectations.

In addition, we would also take other factors such as; code inspection, defect rate, bug severity and a list of many other items, and adjust the schedule that way.

For the customer, it was important that they see how the product progressed over the lifecycle of the effort so we decided on a "STAR" to "Cloud" icon set that would quickly show where we are the software quality level was at any particular time according to the Y-axis on the chart. The wind direction E->W was improving and W<-E was not.

At that time our time was up.

Other Roundtables

Function Point Analysis Methodology

Michael Bragen facilitated a follow up to his January roundtable on Function Point Analysis Methodology at the May SPIN meeting. Topics covered included:

- What is Function Point Analysis?
- How can it be used effectively as the basis for software measurement and process management?
- What are its limitations, costs, and benefits?
- Experiences and use of the International Function Point User Group FPA metric.

Software Engineering Institute Performance Benchmarking Consortium (PBC)

Michael Bragen, Managing Partner of Software Productivity Research, coordinated a roundtable at the June SPIN meeting on the Software Engineering Institute Performance Benchmarking Consortium (PBC), a working group building processes, standards and approaches to collect comparative productivity/quality benchmarking data.

This roundtable introduced the PBC, discussed functional metrics as an integral part of its charter, and how companies (CMMI-oriented or not) can participate and benefit.

SPIN Information

The Boston SPIN is a forum for the free and open exchange of software process improvement experiences and ideas. Meetings are usually held on third Tuesdays, September - June. Boston SPIN welcomes volunteers and sponsors. There is no charge to attend the meetings. Additional information about the Boston SPIN can be found at our Web home page: http://www.boston-spin.org.

For more information about our programs and events contact Michelle Gross, Program Chair, mxgross@comcast.net.

Cancellations (including weather)

Starting at 3pm, we'll notify you via email to the SPIN distribution list, we'll post the notice on the SPIN web page, and we'll send the cancellation announcement to Channel 7 TV.

SPIN Meeting Location

Boston SPIN meetings are held at The MITRE Corporation in Bedford, located at 202 Burlington Road (Route 62), Bedford. Directions can be found on our Web site: http://www.boston-spin.org or on The MITRE Corporation Web site: http://www.mitre.org.

Please be aware that MITRE has advised us that, due to increased security concerns, you will need a Picture ID for admission to the SPIN meetings. We encourage you to leave all carrying bags, backpacks, and briefcases behind (i.e., in your car). Otherwise, you should be prepared to have these opened and inspected upon arrival.

Our Sponsors

A very big 'thank you' goes to all our generous sponsors for supporting our SPIN activities!!

The MITRE Corporation
http://www.mitre.org
The MITRE Corporation provides us with our meeting space including all the audiovisual needs for our speakers.

Chaco Canyon Consulting
http://www.ChacoCanyon.com
Chaco Canyon Consulting works with people in problem-solving organizations who make complex products or sophisticated services that need state-of-the-art teamwork, and with organizations that want to achieve high performance by building stronger relationships among their people.

AccuRev, Inc.
http://www.AccuRev.com
AccuRev provides award-winning, stream-based software configuration management tools that provide both the flexibility and built-in best practices that developers need to manage today's complex, parallel, and distributed software development projects.

Microsoft Technology Centers (MTCs)

http://www.microsoft.com/mtc

The MathWorks
http://www.mathworks.com
The MathWorks, founded in 1984, is the world's leading developer of technical computing software for engineers and scientists in industry, government, and education. With an extensive product set based on MATLAB and Simulink, the MathWorks provides software and services to solve challenging problems and accelerate innovation in automotive, aerospace, communications, instrumentation, process, and other industries. The MathWorks employs more than 1000 people worldwide, with headquaters in Natick, Massachusetts. The MathWorks is currently recruiting company-wide. For career listings, visit http://www.mathworks.com/company/jobs/

Please help SPIN by being a Sponsor

If your organization is interested in sponsoring the Boston SPIN in any way, please contact SPIN Steering Committee member Bruce Taylor at info@workinginunison.com or ask any Steering Committee member for a sponsorship brochure.

Email Lists

To receive Boston SPIN specific notices, send an email to:

Jim Withall, Boston SPIN membership chair, at withall@rcn.com.

To receive Boston SPIN-Plus specific notices, send an email to: spin-plus-request@boston-spin.org, include "subscribe" in the message body.
To post an announcement on the SPIN-Plus list: spin-plus@boston-spin.org

Newsletter Call for Articles

The *In-the-SPIN Newsletter* is always in need of new and interesting articles dealing with process improvement, software development methodologies, project management and other related subjects that may be of interest to our readership. Please send general correspondence or articles that you would like to have considered for publication to:

Judi Brodman, Editor of the In-the-SPIN Newsletter,
brodman@logos-intl.com

Back issues of the *In-the-SPIN* Newsletter can be found on the Boston SPIN Web site: http://www.boston-spin.org/ as well as the guidelines for any articles submitted.

Upcoming Meetings

Date:	Speaker:	Topic:
9/12/2006	Scott Ambler	*"Update on the Unified Process: Something Old, Something New, Something Borrowed, and Definitely Something Blue"*
10/17/2006	TBD	*TBD*
11/14/2006	Steve Rakitin	*"All Software Is Defective - Implications for the Software Industry"*

12/12/2006	TBD	*TBD*
1/16/2007	Johanna Rothman	*"Behind Closed Doors"*
2/20/2007	Robin Goldsmith	*"Avoid Creep-Discover the REAL Require-ments"*
3/20/2007	Naomi Karten	*TBD*
4/17/2007	Ed Yourdon	*"The Internet's Second Wave: Web 2.0"*
5/17/2007	SQCNE panel	*"Exploitable Software: New Security Con-cerns in a Post-9/11 World"*
6/19/2007	TBD	*TBD*

Quote For the Day

"In this age, which believes that there is a short cut to everything, the greatest lesson to be learned is that the most difficult way is, in the long run, the easiest."

- Henry Miller, "The Books in My Life" (1957)

Enjoy the rest of Summer!!!