JUST ENOUGH SCM™

A Concept Whose Time Has Come

Don Peck

A Software Configuration Solutions White Paper
January 31, 2001

Software Configuration Solutions, Inc.
15800 West Bluemound Road
Suite 275
Brookfield, WI 53005
Phone: 262.938.0442
Fax: 262.938.0443

INTRODUCTION

A while back, in a post-project discussion within our company • Software Configuration Solutions (SCS), the topic of exactly what were our customers really happy with and what were they unhappy about came up. The most common kudos we received concerned the experience level and knowledge exhibited by our personnel and the most common complaint, by far, was the length of time it took to implement our suggestions. In examining both those topics, the idea of Just Enough SCM™ began to take shape.

How was SCS implementing software configuration management? What is the discipline like today? How does it really fit into the modern software development environment? To answer these, and other pressing questions, we became real students of not only the SCM discipline, but also the culture that fostered it in the past and affects it today.

To begin, we took a look at the "good old days." Not to reminisce, but to set the stage as to what is salvageable and what needs to be overhauled. Then we spent many hours putting together a new SCM model that today's organizations can reap benefits from without excessively burdening their already overworked staff. Lastly, we became aware that the environments we work in are changing so rapidly, that our new paradigm will have to adapt along with it or face the same complaints a few years hence.

This paper does not advocate the elimination of the SCM process. It explores a method of turning the large, monolithic SCM processes into an object-oriented methodology that provides Just Enough SCM™ processes in a reusable and flexible manner.

THE GOOD OLD DAYS

Most IT personnel today would be as lost in the "computer room" environment of yesteryear as their predecessors would be in their world of rapid application development, Web, e-Commerce, etc. Unfortunately, many of these modern day developers have recognized that majority of the processes that attempt to support their new world are relics of their predecessors and have not kept up with their environment. They are discovering that the overhead is too much; they don't have the time or the desire to become part of the "P" world of Policies, Processes, and Procedures. Software configuration management, in many of these organizations, is included in this overhead. Interestingly, many enterprises are attempting to get things under control by "tidying things up" or imposing more restrictive processes and exerting more control.

The "P" World

My grandson Zachary, age 3, walks into my house, says, "Hi Pop, I want to play the compooter." He sits down, boots the system, chooses Shanghai®, takes out the CD from the jewel case, inserts it, clicks Start Game, chooses the Tile Set he wants to play, and begins to match tiles in a modern day Mah-Jongg game; with no assistance from anybody. As a matter of fact, if I foolishly try to tell him something he already knows, I risk a look that would strike terror in Attila the Hun.

The International Society for Technology in Education states that by the end of the 2nd grade students should be able to:

- Use a computer keyboard, mouse and printer.
- Use accurate terminology to describe computer technology.
- Create multimedia projects suitable for their age level.

By the time they leave 5th grade they should:

- Use PowerPoint, the Web, digital cameras, and scanners to put together presentations.
- Select the appropriate technology for specific tasks and problems.

And, by the time they leave high school, their capabilities will include:

- Identifying the capabilities of available and emerging technologies.
- Assess how the above technologies might be used in workplaces,
- Collaborate with peers to create a database.

Do these sound like the future workers who are going to be willing to use 1970's processes or even take the time to read them? We found that we didn't have to project into the future. Today's workers are already refusing.

Figure 1 shows the classic process being made up of Inputs, the Process itself, and Outputs. In documenting how to implement

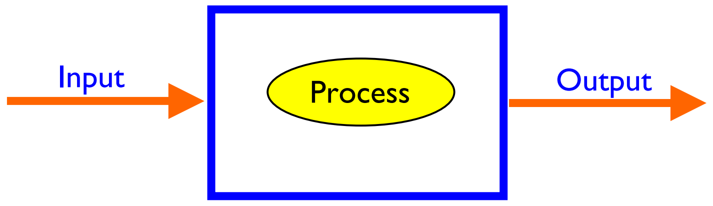


Figure 1

a process, much time is spent on format, partitions, sections, terms, and etc. but seemingly little time on its usefulness. The modern day user doesn't want to dig through pages of SCM Management, Organization Structure, Scope, Responsibilities, … to find out how to checkout a component. They are not interested in knowing that what they are checking out is called a configuration item and it uniquely identifies, names, and describes the documented physical and functional characteristics of the code, specifications, design, and data elements to be controlled by the project they are working on.

So, what are they interested in? They are interested in something that tells them how to do the checkout they have to do and don't know how to do it. This something may be:

1. A step-by-step instruction.
2. A form.
3. A checklist.

It will not be an onerous process steeped in tradition to fulfill a requirement for a standard from another world or another time.

The new "P" could look very much like Figure 2.

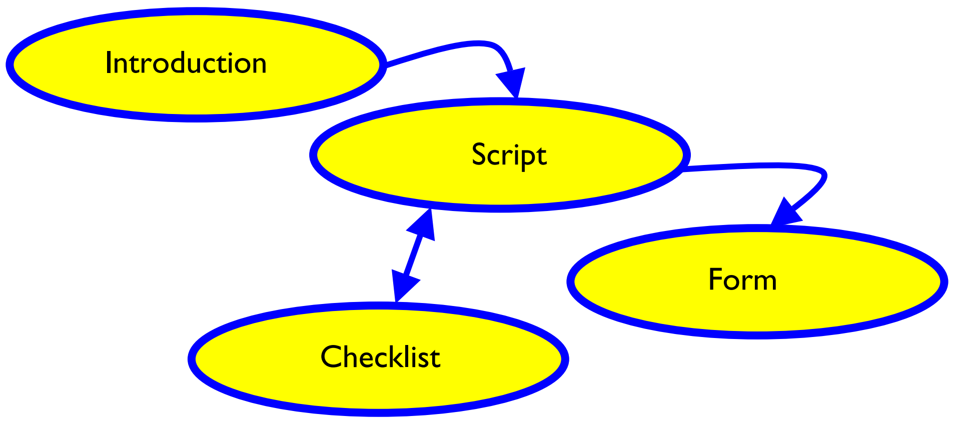


Figure 2

The Introduction would briefly describe the purpose of the Script and what to do with the Form and/or Checklist, the Script describes step-by-step how to do something, the Form could be an output of the Script, and the Checklist makes sure that the key steps in the Script are done. **Just Enough** to help them do their job.

The four parts of Figure 2 could be objects thus, it may have a completely different look, but it won't take a week to change it. I will cover more on this later in this paper.

The Standard Way Of Doing Things

The status quo of the "P" World has its biggest supporters in the "modern" standards that populate the shelves of our clients. Bluntly put, the majority of them need a serious dose of reality. Many of them fit nicely in the "good old days" but have serious shortcomings in today's environment. Deep analysis of our clients reveals that the majority of their complaints center around the rigidity and heavy-handedness of the standards and there seemingly reverence to details that don't fit their situation.

Every client interviewed yearned for a criterion to follow but one that makes sense to the way that they do business, not as a yardstick for auditors or assessors to follow. In the eighteen to twenty-four months that most of the standards are quoting to implement, the majority of the client's workers may very well have changed employment or careers.

JUST ENOUGH SCM TODAY

It would be blatantly unfair to blame all the ills of the software industry, including software configuration management, on processes. In retrospect, the purposes of the processes we examined are still as appropriate today as they were twenty years ago. The magic comes in adapting and implementing them in an acceptable manner.

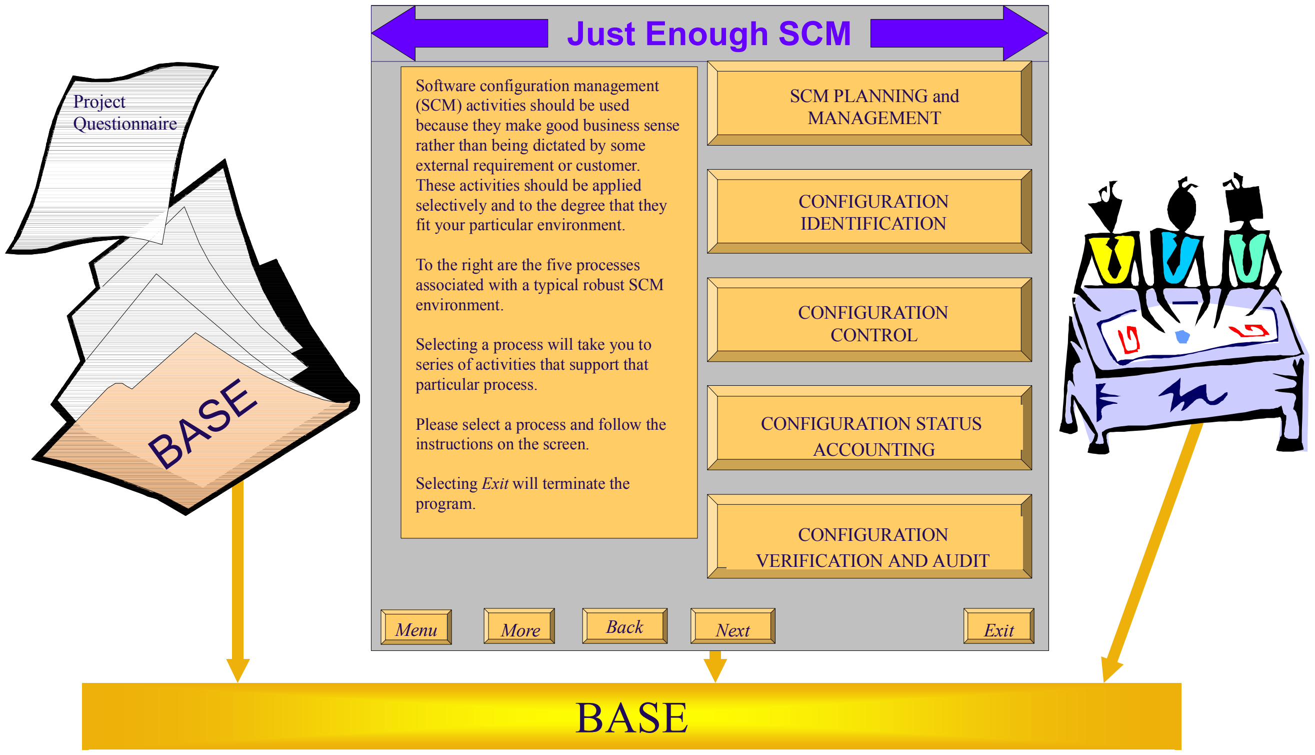
BASE

When Software Configuration Solutions is contacted about providing a solution, we like to know exactly what situation the client is in and exactly where they want to go. Sounds reasonable. The devil is in the details. First of all, it is very expensive and time consuming to determine "exactly." Secondly, clients often do not know "where they want to go." Don't get me wrong, this is not saying that we advocate walking into a situation completely blind. We want to know the local situation so we can sit down with the client to plot a strategy and a direction that makes sense. However, we studied what did we really need to know initially.

We discovered that much of what we "assessed" was recorded and forgotten or was a lot of background information that had little bearing on the problem(s) at hand. Many of our clients balked at a two to three week assessment to uncover the very problem they were hiring us to correct.

We devised a "Just Enough" assessment that we called • Brisk Assessment of Software Environment or *BASE* to obtain the current software environment that our client was presently working in. BASE uses a sampling technique of questionnaires to target trouble and propritary prioritization techniques to study the real problems instead of the whole situation.

As can be seen in Figure 3, the BASE process begins with some tailored questionnaires sent to the client. When the questionnaires are returned, the results are analyzed for completeness and to tailor a site visit. A site visit is conducted (Usually three or less days) and upon return a BASE Assessment Report is generated stating the Strengths, Weaknesses, Opportunities for Improvement and SCS's Prioritization Recommendation. The client and SCS then prioritize and weigh the client's needs and resources to arrive at the best agreed upon actions.


Software configuration management (SCM) activities should be used because they make good business sense rather than being dictated by some external requirement or customer. These activities should be applied selectively and to the degree that they fit your particular environment.

To the right are the five processes associated with a typical robust SCM environment.

Selecting a process will take you to series of activities that support that particular process.

Please select a process and follow the instructions on the screen.

Selecting *Exit* will terminate the program.

Project Questionnaire
BASE
Just Enough SCM
SCM PLANNING and MANAGEMENT
CONFIGURATION IDENTIFICATION
CONFIGURATION CONTROL
CONFIGURATION STATUS ACCOUNTING
CONFIGURATION VERIFICATION AND AUDIT
Menu
More
Back
Next
Exit
BASE

Figure 3

BASE has proven to be very successful. Clients rave about our ability to gather and analyze data in a few short days instead of the two to three weeks that we use to spend.

Element Objects

The next aspect of adapting processes to please our clients came to light when we began to evaluate what our clients were really asking for. Software Configuration Management was too broad a term to fit most of our clients. Very seldom did we find a client that wanted or needed everything. Usually the BASE showed that they were quite weak in one or two partcular areas. What worked, worked well was Just Enough SCM to fix what didn't. Time and time again, what was working fell into the same categories:

1. A simple step-by-step way of doing something.
2. An easy to follow form.
3. A checklist to keep them on track.

What didn't work was something that was a totally new situation being managed by an obsolete process or manged by a process that was more complex and/or cumbersome than the task. Another characteristic that began to emerge was the similarities of the steps, forms, and checksheets between clients. Using our experience, we found that introducing a step-by-step procedure, a form, or a checklist that was successful for one client produced rapid and greatly appreciated results for similar clients.

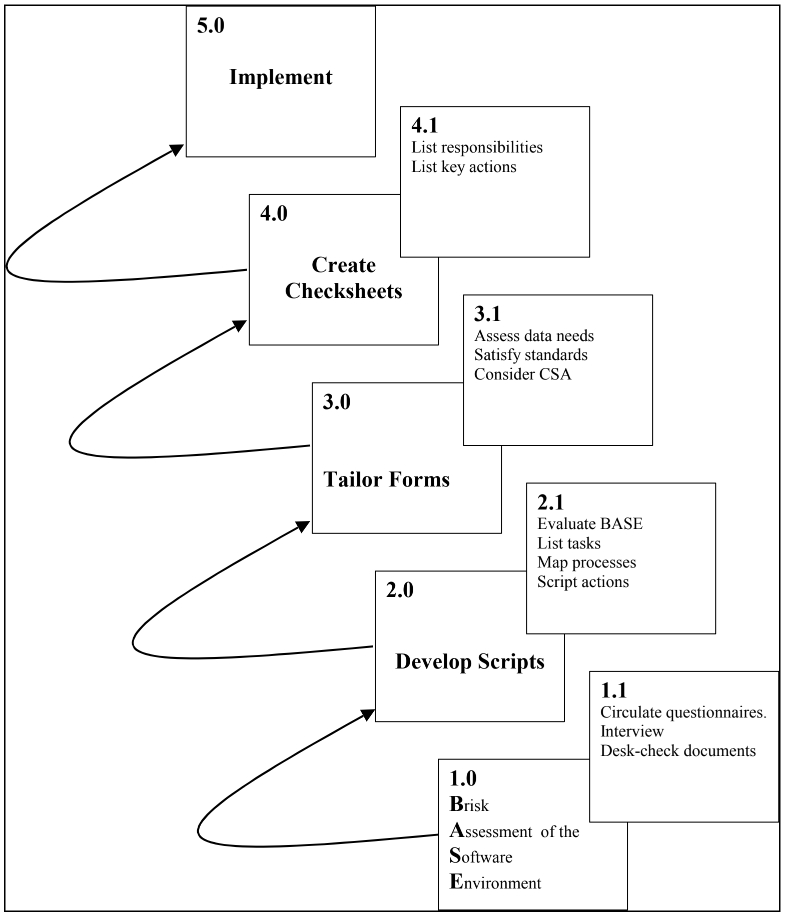


Figure 4

We formalized and catergorized the step-by-step procedures into self-contained scripts and began collecting input and output forms and checksheets. Converting them to template format and assigning them a metadata locator, we had begun the foundation of Just Enough SCM. We call these tailorable plug-and-play ingredients, Element Objects. Figure 4 depicts today's Just Enough SCM process.

JESCM Objects

If BASE determined that there was a weakness in your SCM process, we would combine all the needed Element Objects to accomplish an SCM procedure. This combination results in a Just Enough SCM object we call a JESCM Object.

JESCM Objects may fulfill the need for such things as:

Configuration Identification
Version Management
Configuration Status Accounting
Software Configuration Management Planning

OBJECT ORIENTED BASICS

As exciting as today's environment is, it promises to become even more so. Software Development vendors are already selling self-contained software components that can be used to build extremely complex applications with little or no programming expertise. We know that this concept is just around the corner for processes also. As a matter of fact, we have already begun with the planning of true object-oriented Just Enough SCM.

In order to understand where we are headed in Just Enough SCM, we need to take a look at the concept of object-oriented (o-o). An in depth look at o-o software is way beyond the scope of this paper. Besides, this is a paper on Just Enough SCM, so I am going to give you a Just Enough explanation of o-o.

You are, or should be, familiar with LEGO® bricks. If not, study Figure 5 intently. These bricks fit snugly with each other and create bigger things. Each LEGO® brick is thought of as an object and the thing you create by putting them together is an object.

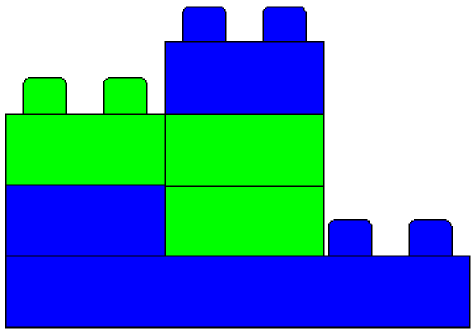


Figure 5

Objects are an abstract way of viewing real life things. If I was a great architect and wanted to simulate what I could build with LEGO® bricks, I could write an object that simulated a LEGO® brick and start building. If I wanted to add pulleys (another object) and wheels (and another), I would write the software that simulated these objects. Note, that I would be a terrible architect if I somehow connected my wheel object to my logo object, because they are two distinctly different things. If I changed my LEGO® bricks to Lincoln Logs®, it shouldn't change in anyway the characteristics of my wheels or my pulleys.

Classes

It isn't asking too much to ask for you to let your imagination run a bit and see that object-oriented programming is modeled on the premise that real life things are just objects that are made up of many kinds of smaller objects. If everything in life was made up of LEGO® bricks and the pieces to use with them, it would be simple but, I fear a bit tedious. When you were reading the explanation of objects, I mentioned changing the LEGO® bricks to Lincoln Logs®. I could have just as well said Tinker Toys® or Erector Set®. All these building block objects belong to a *class* of model building objects and it is to this class that I would write my program.

Inheritance

The final concept that I want to briefly describe to you is the concept of inheritance. If a class exists that contains many of the characteristics (attributes) of a new similar class I want to use, then why would I have to write the new class from scratch. The answer is, you don't. All you have to do is create the new class as a subclass of the existing and the principle of inheritance says that the new subclass inherits the attributes of the existing one. This promotes the concept of reuse and is a key to our later discussion.

SCM CLASSES

By now, you may be thinking; "This is really fascinating and the next time my five year old sits down with his blocks, I will tell him all about it. But, what does this have to do with the future of Just Enough SCM?" Actually, Everything!

Let's leave the architect metaphor and look at classes that you may be more familiar with. I am sure that in your experience you have run across the statement that software includes everything that is put down on some type of media. It includes everything from the builds you make, to the scripts you use to run the build, to the procedure that explains how to make your builds, to … There could be a case for making "Software" a Class and everything you read, write, or code, objects under that class. Indeed, in some process simulations I have seen and used, this has been done. However, in this discussion I would like to focus on another familiar class — documents.

Document Class

Think off all the documents in your organization. When you have recovered from that, narrow your thoughts to just the documents that it takes, or is going to take, to run your software development process (SDP). Further narrowing it down, picture the documents that are part of just the SCM process. Collect four of them. Open them up and see how many Introductions, Scopes, References, Roles, Responsibilities, etc. that you see. Now think back up the line to how many Introductions, Scopes, References, Roles, Responsibilities, etc there are in the rest of the SCM, the SDP, and the whole rest of the organization. Every time a new "P" is written, there needs to be a whole new set Introductions, Scopes, References, Roles, Responsibilities, etc written. If a class existed that included Introductions, Scopes, References, Roles, Responsibilities, etc as attributes, your new "P" could inherit them from that class.

But, you say that each "P" would have different Introductions, Scopes, References, Roles, and Responsibilities. Absolutely, and we will just assign unique values to them when we write the o-o program that writes them. The neat thing is that in many cases, not all of the attributes are needed. However, if you really must have certain attributes, inherit them from the Parent Class, don't rewrite them.

So what does a Document Class Look like?

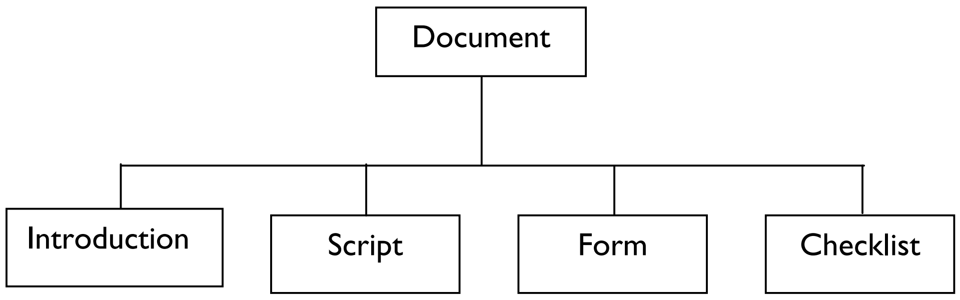


Figure 6

Once you have described the parent class (Document) then its subclasses inherit all its attributes (Page number, footer, …) any instance of that subclass is an object. Say again! Think of the subclass as a blueprint and the object as the actual thing. The document objects shown in Figure 6 could be any of a myriad of documents and may use or not use any of the characteristics of the Parent Document Class abstract.

Figure 6 shows four parts to the Just Enough SCM model that I have chosen to make Classes. If a unique document subclass should appear, then it would be added as another subclasses under Document.

Introduction Class

The introduction class is a pure document type. It would inherit attributes such as font, style, title, and page format from the parent document class. It would add such attributes as Objective, Scope, Roles, References, and Responsibilities to make itself a unique class.

Script Class

The script class is what most people are looking for when you mention Just Enough. Scripts tell the reader exactly how to do something, nothing more and nothing less. It too would inherit the font, style, title, and page format attributes and may add at least three attributes: Step – Action – Expected Outcome to form its class. Your script class could be as simple as that. The objects you make from that class will include those attributes with unique values assigned to each.

A note on script objects—Do not in today's script templates or tomorrow's element object scripts make the script all encompassing. In other words, don't write a script on Version Management. Write a script to Check-in, another to Checkout, another on how to Recall a Version, and etc.

Form Class

The Form Class, along with the yet to be mentioned Checklist Object, are rapidly becoming the most "object" like in our Just Enough SCM. SCM forms such as Change Request, Handover, and Code Bill-of-Materials have been around quite some time and

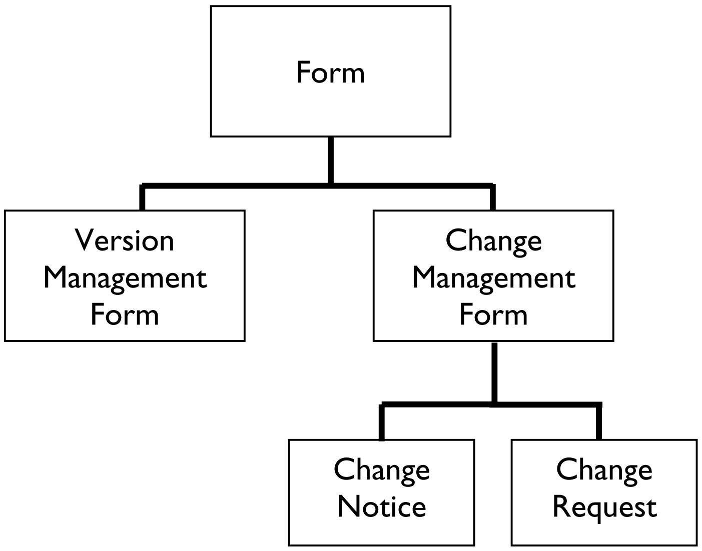


Figure 7

the characteristics of forms in general are quite well understood. Forms and Checklists adapt themselves readily to electronic processing better than other documents, thus the possibility of true o-o programming techniques can be applied more easily.

In Figure 7 the attributes of Form could be:

- Columns
- Rows

The attributes of Change Management Form:

- Issue
- Severity
- Priority

The attributes of Change Request:

- Submit Date
- Change Number

Programatically, I could assign the Change Request's or Change Notice's Font by the inherited Document Font attribute but the Change Request may have status attributes that are not part of the Form or Change Management Form Class. Thus, I couldn't assign them to the Change Notice. If I had the need for another Change Management Form, I would describe another subclass like Engineering Change Notice and would only have to describe the attributes that make it unique.

Checklist Class

The last class in the Just Enough SCM o-o kit, is the Checklist Class. Checklists are used to make sure that key milestones in a script are done or to ensure that key points are reviewed in inspections, audits, or reviews.

As mentioned above, checklists, like forms, are quite stable and lend themselves to automation. By now, you should be able to visualize the beginnings of a Checklist Class Hiearchy.

SUMMARY

We have journeyed from the “good old days” of the computer room to the hetic world of today’s e-commerce and even took a peek into the future. Along the way we looked at some of reasons why organizations are shying away from Software Configuration Management exactly at the time when they should be embracing it. We finished up by looking at Software Configuration Solution's Trademark Just Enough SCM techniques to implement these crucial processes in a fast and proven way.

Modernizing your software configuration management processes need not be a long and drawn out project. SCM is vital to your software development and the sooner your people see it as an aid instead of a hinderance, the better. Using the techniques we have touched on is this paper can make that a reality.

For further information please contact:

SOFTWARE CONFIGURATION SOLUTIONS, INC.

15800 West Bluemound Road, Suite 275
Brookfield, WI 53005
ph. 262.938.0442 fax 262.938.0443
toll free 888.838.0442
www.softconfig.com

FURTHER INFORMATION

The following Software Configuration Solutions, Inc. white papers are available at no charge.

★ Just Enough Software Configuration Management™
★ A Step Towards Software Excellence: SCM

Copyright and Permission to Reproduce

About the Author

Don Peck, M.B.A. has over 30 years experience in the technical and management aspects of software configuration management, software quality assurance, project management, and software development for emerging and established companies. In software configuration management Don has directed all aspects of this critical discipline from assessments to tool evaluations to consulting for a total implementation effort. As a software quality manager, he maintains certification as an ISO 9000 Lead Assessor as sanctioned by the British Standards Institute. Don maintains a position as a Software Chair for Association for Configuration and Data Management, is a member of the American Society for Quality and American Society for Training and Development. Since 1998 he has been a valuable part of the Software Configuration Solutions, Inc. team working as a SCM consultant with Fortune 500 corporations around the globe.

About Software Configuration Solutions

Software Configuration Solutions, Inc. (SCS) offers the creation of high quality software with bottom line results to Fortune 500 corporations around the globe looking to maximize their environments. Creating such environments requires blending tools, talent and training to establish an infrastructure receptive to change. With collectively more than 100 years of seasoned SCM experience, SCS consultants understand what it takes to maximize development. SCS uses solid processes, enabled by tools and supported by knowledgeable staff to improve productivity and reduce costs allowing our clients **FREEDOM TO FOCUS**. More information about SCS can be found at WWW.SOFTCONFIG.COM or by calling toll free 888.838.0442