

THE VALUE OF PROCESS

CREATING AN EFFECTIVE DEVELOPMENT ENVIRONMENT

PRINTED: WEDNESDAY, APRIL 05, 2006

TABLE OF CONTENTS

INTRODUCTION..... 1

CHANGING NATURE OF DEVELOPMENT..... 2

Complexity of Parts..... 2

Software Development is Not Engineering..... 3

USING SOFTWARE FOR SOFTWARE DEVELOPMENT..... 5

Avoiding Out-of-Band Processes.....5

Low-Level Documentation..... 6

High-Level Documentation..... 6

Requirements Documentation..... 7

CHANGE AND PROBLEM MANAGEMENT.....9

Version Management.....9

CONCLUSION..... 11

THE VALUE OF PROCESS

CREATING AN EFFECTIVE DEVELOPMENT ENVIRONMENT

INTRODUCTION

One of the difficult choices in the management of a software development project is the role of the processes that support the development effort. This discussion centers on one key principle – that of the customer's point-of-view. The only thing that the customer really cares about is the delivered software. All of the other byproducts of development don't matter to the customer. It's not that they don't matter. It's just that they don't matter to the customer.

It's like buying a car. We care about the car – the way it looks, the way it handles and what it costs at the very least. At a more subconscious level, we care that it functions properly and is of high quality. *That's* where process adds value. Process is necessary to ensure that the software product produced is of high quality and functions properly, but we can't forget the fundamental notion that the only thing that really matters to the customer is the software product. Therefore, every software development process that is put in place, must add value to the delivery of the software product.

Software development is closely tied to the business. Either the business is producing software for direct sale or to support an internal business function or service. As a result, managing the cost of a software development process is important. Since process takes time, and therefore cost, the challenge to decision makers is to implement just the right amount of process to accomplish the business objectives, but not one bit more.

That's a considerable challenge. Many of the issues are very technical in nature. Software development itself continues to evolve and that constantly changes the rules. Every development team has different dynamics. A team with a very strong technical lead and good programmers needs less process than a team of programmers with typical skills. Unless there is a strong architectural foundation for software, a team of contract programmers will likely need more process than a team of developers that is part of a corporate culture.

The most effective approach to the software development process is to have a small core of mandatory processes and a lot of processes that can be interjected or removed from the development process at the discretion of those managing the project. This approach to managing a project places a greater burden on the management team, but this approach increases the opportunity for a highly-quality, highly-functional, cost-effective development team.

THE VALUE OF PROCESS

CREATING AN EFFECTIVE DEVELOPMENT ENVIRONMENT

CHANGING NATURE OF DEVELOPMENT

In the last quarter-century, things have changed a lot. Perhaps the most profound change in IT is the staggering increase in complexity. The good news is that there has been a complimentary increase in the ability to address that software complexity, both in the tools used to produce software and in the tools to manage the development process.

COMPLEXITY OF PARTS

As development moved from procedural languages like PL/I, COBOL and C in the 1970's and 1980's to object-oriented languages such as C++, Java and .NET-based languages in the 1990's, there was a corresponding shift from a small number of monolithic parts to a large number of granular parts. A software project in the 1980's that was six large modules in COBOL might now be a few hundred objects that leverage a dozen class libraries, each of which may have hundreds or thousands of parts.

The benefit of this approach is an substantially-increased opportunity for reuse, which lowers development cost because there is less new work that needs to be done. It also increases quality because there is less new code to test. Since the turn of the century, new approaches to component and frameworks-based development have significantly improve reuse opportunities over plain object development, but have further increased the complexity of build and deployment. Instead of managing the versioning of a few large modules, we have to manage the inter-relationships between the various versions of code we are reusing. In spite of these issues, the “complexity of parts” problem is a much better problem to deal with than the “monolithic complexity” that was a product of the 1980s.

The complexity of parts problem affects the software development processes. The first effect is the need to test software at a granular level in a way that is automated. There are tools that allow for the construction of unit tests within a framework that enables them to be tested in an automated fashion. I am involved in a software development project for which there are dozens of granular tests against a code-base of around 100,000 lines. Every test is run at least once per day. If one of the developers makes a change that had an unintended ripple effect in another part of the code, we have a good chance of detecting that problem within 24 hours. That allows us to identify and resolve the problem more easily than discovering it during formal acceptance testing weeks later.

THE VALUE OF PROCESS

CREATING AN EFFECTIVE DEVELOPMENT ENVIRONMENT

If your processes still talk about “modules”, there is a good chance that the processes don't adequately address the complexity of parts issue.

SOFTWARE DEVELOPMENT IS NOT ENGINEERING¹

The process of software development borrows a lot of concepts and terminology from engineering. There are aspects of software development that are similar to various engineering disciplines, but software development is not an engineering discipline in the same way that other engineering disciplines like construction engineering or electrical engineering are. Software development can be managed that way, but it shouldn't be.

In the early 1980's, there was a movement to interject an engineering discipline into software development. A lot of good ideas came out of that effort that are still in use today, but a strict engineering discipline in software development environment proved to be unworkable.

Most engineering efforts are oriented toward the mass production of an object for resale. In that environment, there is a need for rigorous specifications and design since the product will be mass-produced and the cost of design will be amortized across thousands or millions of units produced. Most enterprise software development is focused on producing a single product.

If we look at engineering projects that produce a very small number of products, we can more closely approximate software development. An engineering firm that specializes in bridge-building or stadium construction appears more like enterprise software development. When that model was applied to software development, we got a rigorous approach to upfront design specifications. Those specifications were manufactured into software by developers. That is essentially the waterfall development process and the corresponding silver bullet of the 1980's – Computer Assisted Software Engineering (CASE) tools.

While many successful software projects have been implemented with that approach, the model is more expensive and time-consuming than is necessary for software development for one important reason. Software is soft. It can be cost-effective to change how

¹ The term “software engineering” is still used a lot. The Software Engineering Institute is doing some great work in defining software processes for real-world, contemporary projects. We have to note that SEI was founded in the 1980s when “software engineering” was the new buzz. The name stuck, but approach has changed into something distinct from traditional engineering.

THE VALUE OF PROCESS

CREATING AN EFFECTIVE DEVELOPMENT ENVIRONMENT

the product works while it is being manufactured. The best processes leverage the malleability of software in a way that engineering for the physical world can not.

THE VALUE OF PROCESS

CREATING AN EFFECTIVE DEVELOPMENT ENVIRONMENT

USING SOFTWARE FOR SOFTWARE DEVELOPMENT

One thing that we have learned from the various engineering disciplines is that it is often cost-effective to manufacture tools to assist in the manufacture the product. As mentioned earlier, there are frameworks for unit testing that can be used to automate that tedious process. There are also tools that manage the complexity of building and deploying software artifacts. Some of these also reduce the complexity of version management as part of the build/deploy process.

The most effective software development teams take the time to invest in software that assists in the software development process. Since these tools all fall under all non-functional requirements, it's easy to overlook the value that these tools bring to producing high quality software that meets the functional requirements.

AVOIDING OUT-OF-BAND PROCESSES

It's almost universally true that developers dislike process. Well, that's just tough. Processes are important for quality and management of a project. Most developers will begrudgingly admit that is true. The best way to get developers to comply with the processes is to implement processes that are easily integrated with the way in which they work.

Most software developers live in their integrated development environment (IDE). For .NET developers, that is most likely Microsoft Visual Studio. For Java developers, the market is a little more fragmented, but by most estimates the market leader is Eclipse. The processes that integrate well in the development environment will most likely have the greatest degree of compliance² with the least amount of cost.

2 ... and the least amount of complaints. Developers are smart people. They know how to “work the system.” Software development projects work better developers have a “system that works.”

THE VALUE OF PROCESS

CREATING AN EFFECTIVE DEVELOPMENT ENVIRONMENT

LOW-LEVEL DOCUMENTATION

The best documentation on the internals of a system³ is going to be documentation that is maintained by the developer. Updating external documentation such as a word processing document is an out-of-band process. It is likely that over time, the external documentation will drift from the implementation with the code. This can happen for multiple reasons, but the most common is that time-to-market pressures delay the updating of the documentation until some point in the future. By then, the changes either don't get done at all or much of the detail is lost.

Of course, we can put a process in place that ensures that the developers update the external documentation before the code is placed into production. We can have another group ensure that the processes are being followed, but that is a rather antagonistic approach to software development⁴. There is a better way. Contemporary languages such as Java have a process for including the low-level functional workings of the code within the source code itself. Detailed, hyperlinked documents (“javadocs”) can be created automatically from the source code. The developer still has to enter the information in the source code, but it is substantively different because it is part of the natural flow of the development process. If a project wants to put a stake in the ground for effective documentation, the use of good javadocs should be a requirement of the development process. Instill in the developers a core value for providing accurate javadocs. When this is part of a larger continuous integration process, peer pressure will work well to ensure that the developers are doing this because there is a lot of visibility within the team to the source code and the documentation is part of the source code.

Some of the .NET languages, notably C# have open-source tools that handle documentation similar to javadoc, but there is nothing that is currently supported by Microsoft.

HIGH-LEVEL DOCUMENTATION

Most of the high-level documentation is produced at the requirements gathering phase. This covers the function of the software at a high level of abstraction. If a good understanding of the needs of the software is gathered before development starts, the high level design should remain largely unchanged throughout the life of the project. As a result, there shouldn't be a lot of effort required to maintain this documentation since it should be fairly static.

3 ... as opposed to end-user documentation

4 The floggings will continue until morale improves.

THE VALUE OF PROCESS

CREATING AN EFFECTIVE DEVELOPMENT ENVIRONMENT

REQUIREMENTS DOCUMENTATION

There is a broad range of requirements documentation that fall between the very granular documentation afforded by tools like javadoc and the high level documentation that describes the shape of the software at an architectural level. Requirements documentation is produced at the initial requirements gathering phase and form the basis of the first iteration of code.

The current state-of-the-art in software development centers on an iterative process. Because software is so malleable, we have an opportunity deliver software that approximates what the customer tried to articulate as a need and get feedback to further refine the implementation. We can't do that with bridges or stadiums, but we can do that with enterprise software. A side-effect of the iterative process is that the initial requirements documentation gets out of date very quickly. To reiterate the guiding theme, all of our processes should add value to the production of running software that is functionally correct and of high quality. Every developer knows that requirements documentation tends to get stale very quickly, especially in an iterative development environment. When we consider putting a process in place that ensures these documents remain updated, we need to consider for whom we are doing this work. The original requirements documentation was for the developers. Once there is a code base in place, the developers will look to the code to determine what the code does, not the functional requirements documents.

We can put a process in place that ensures that the mid-level requirements documentation is updated before code is released into production, but this is cost-inefficient and unnecessary because of advances in integrated software development tools. Once again, there are attributes of software that are different that those of the physical world and we can leverage that to our advantage.

The most common way to represent functional requirements are UML⁵ diagrams. They can assist the developers in the understanding of how software is constructed. Many of the UML diagrams can be reverse-engineered from existing source code, effectively producing documentation about the functional aspects of the software. Unlike documentation maintained in an out-of-band process, this documentation is accurate by definition, because it was generated from the source. Quite often, the IDEs have sufficient sophistication so that generated documentation such as UML diagrams are unnecessary. Source code is automatically linked in inheritance hierarchies and call hierarchies so that developers can rapidly gain an understanding of how the software functions.

5 UML stands for “Unified Modeling Language” because the two market leaders for modeling software function decided to take the best of each and create a single way to express design concepts

THE VALUE OF PROCESS

CREATING AN EFFECTIVE DEVELOPMENT ENVIRONMENT

The other development tool that can be an effective as a partial replacement for out-of-band requirements documentation are the test cases that are codified in a testing framework. If there are software test cases that model the functional requirements, then those requirements are codified and are readily accessible to the developer to assist in the understanding of how components should work. Unlike out-of-band documentation, these tests are “live” and continue to add value to the development process throughout the life of the code by ensuring that functionality is not regressed as problems are corrected and requirements adapt to the changing business needs.

Too often, there is too much emphasis on updating documentation that has little or no audience when other forms of documentation and test cases encapsulated in code are more useful and less costly.

THE VALUE OF PROCESS

CREATING AN EFFECTIVE DEVELOPMENT ENVIRONMENT

CHANGE AND PROBLEM MANAGEMENT

Change and problem management is a core process that is a requirement for every software development effort. Unfortunately, it is also an out-of-band process to the developers. Although required, there is still a significant degree of latitude in how to implement the change process. Some enterprises have implemented change and problem management for general oversight and assisting with priorities and scheduling of changes and fixes. Other enterprises have deeply embedded the change/problem process into the development process so that they are an approval and compliance committee.

Like many other business processes, the best software development processes adapt to the situation instead of taking a dogmatic approach. That presents a challenge to the decision-makers, since there are more decisions to be made and more nuance to the reasons behind the decisions. At the other end of the spectrum are organizations that value process over product. In one situation, a one line code change⁶ required the full heavy-weight change management and approval process.

VERSION MANAGEMENT

Part of the debate within the change and problem area is the frequency of the build, full test and deployment to production. Even with a considerable amount of automation, the production deployment cycle is a heavyweight process.

One of the key questions that needs to be managed is whether certain changes or fixes are in production. Until recently, this has been an out-of-band process for the developers that was accomplished through the source code management tool and conventions for tags or comments. As a result, the answer to that question hasn't always been easy to determine. Contemporary languages such as Java⁷ have added a new concept in programming languages called metadata. This allows information to be added to the source code that is not part of the algorithmic part of the language. This metadata is most frequently used by container-managed environments or for compiler "hints." We can use this facility to track changes and fixes to the code in a manner that is very much a part of the natural process for software development. Annotation such as the following:

6 Literally. No hyperbole there.

7 Although Java is 10 years old, these feature became available only in the more recent release.

THE VALUE OF PROCESS

CREATING AN EFFECTIVE DEVELOPMENT ENVIRONMENT

```
@TrackingNumber("657")
```

indicate that this code was modified as a result of the problem or change number “657”. This is more than a special comment. This can be compiled into the binary runtime such that a tool can automatically tell us what changes and fixes have been applied to the code currently running in production. If the change management tool is sufficiently sophisticated, we could use these annotations to automatically generate release notes from the text in the change management system. As long as we put the stake in the ground and mandate that the developers do this annotation, we have the potential to increase the efficacy in determining what fixes and changes have been implemented.

This is but one example of using the evolving capabilities of languages to interject process in a way that is a natural part of development. In a continuous integration development environment, there is a lot of visibility to the changing source code. Peer pressure can be effective in ensuring that the developers use proper annotations as they change the code.

Developers should be thought leaders in how to use the facilities in modern languages to integrate with processes such as change and version management. Since change and problem management can be a time-intensive process, managers need to be willing to continually fine tune the process to the situation instead of plodding down the process path.

THE VALUE OF PROCESS

CREATING AN EFFECTIVE DEVELOPMENT ENVIRONMENT

CONCLUSION

There are a few notions about software development processes that are generally agreed upon. The first is that some processes are absolutely necessary. The second is that developers tend to resist anything that is outside what they want to do the most – produce code. The most effective management approach is to interject the minimal number of processes that are needed to meet the business goals. That optimal point will change throughout the project and will vary from team to team.

It been my experience that too often, the process becomes the deliverable. That is, the value of many of the artifacts of the development process become as important to the organization as the software artifact. When that happens within an organization, it is very likely that the productivity of the software development groups is going to suffer unnecessarily.

The teams that produce the most are those with talented technicians who are guided by a management team that understands the minimal set of processes are needed to produce the results that the business demands. These processes are implemented in a manner that is natural to the developers working on the source code. In the best environment, the management team and development teams work together to ensure that the processes add sufficient value to the delivery of quality, functionally-correct software.