EXECUTIVE BRIEFING SERIES

WHITE PAPER

Volume 4

How to Get Custom Software to Market in 8 Weeks or Less:

Applying the principles of Extreme Programming for business success

Geneer®

Building Software. Building Business.SM

©2001 Geneer Corporation

VOLUME 4

How to Get Custom Software to Market in 8 Weeks or Less:

Applying the principles of Extreme Programming for business success

Contents:

© 2001 Geneer Corporation.

INTRODUCTION

How many times have you specified a software system, using whatever methodology your development team requires, only to find that the delivered system falls considerably short from your original idea? This frequent occurrence is not because developers are dumb, ornery, or contrary. It happens because during the course of the project, decisions — both small and large — are made to overcome technical hurdles or figure out exactly what you meant by that requirement. These decisions can cause a project to drift away from what you originally intended. And, you yourself may have refined your vision due to market pressures or improved understanding during the course of the project.

Often, the result is software that is on time, on budget, and almost of no use. Much of the software developed today is rendered inadequate either because the customer, economy or industry changed direction before the product was finished or because the end product was not what the customer ultimately needed or specified.

In most standard development methodologies, developers are driven by an overall design that was completed at the beginning of a project and by the deadlines resulting from that design. At the beginning of a project there are often many unknowns. This can create a gap between what was intended at the beginning of a project (designed when everyone was new to the project) and the final product, which is the result of experience gained through both good and bad choices during the product's life cycle.

On the other hand, as the business owner of the project, you really don't want unquestioning obedience to a specification and a design created at the project's start. That would be like creating a turn-by-turn map between here and Calgary and asking your driver to execute it exactly; turning the steering wheel only when the plan indicates. Obviously, in the real world, your driver would need to take into account curves in the road, other traffic, and the fact that you can't turn right exactly 1.7 miles down the highway and not expect to end up in a ditch. Efficient driving, like efficient software development, requires ongoing and short course adjustments. To get to your destination, your driver needs to account for all the obstacles he or she encounters — as well as the surprises like road closings that pop up along the way. You want to allow the driver to deviate from the plan yet still arrive safely at the destination. You also want a driver that can respond if you decide you'd rather go to Toronto.

The same is true with system development. What is needed is not stricter following of process, but more freedom in refining the target and the methods used to get there. This is particularly important with a risk-intensive project. In fact, a Standish Group survey of 8,000 software projects found that the most common causes of

© 2001 Geneer Corporation

project failure were risks that probably were not even considered at project inception: incomplete requirements, lack of user involvement, lack of resources, unrealistic expectations, lack of executive support, and changing requirements[1]. A KPMG Peat Marwick study found that 48 percent of projects experienced problems due to bad planning and estimating.[2] Historically, in software development, rationalization, unrealistic hope and optimism, and inexperience have torpedoed the quality of estimates and the resulting development plans. Once the first domino falls, the results are fairly inevitable: The team misses the mark.

To attempt to rescue projects in trouble, software managers have hired more people, downsized the features of the software, let the schedule slide, and tried to improve productivity. To try to prevent problems, developers have tried a succession of "savior" methodologies and tools as they come up, such as RAD, JAD, UML, frameworks, and component reuse. Yet still the problem persists.

© 2001 Geneer Corporation.

WRITING SOFTWARE RIGHT

As we move into an era where computers and devices are pervasive to the point of being invisible, the development roadblocks are shifting from hardware and networking to the underlying bottleneck inherent in technological change: the lack of a predictable, repeatable, rational process for software development. Noted software design guru Bruce Powel Douglass puts the problem this way:

> What prevents the pervasive computing era from becoming a viable, flourishing ecosystem instead of a fantasy? Simply put, the massive amounts of software it takes to create individual devices, link them together, and provide the distributed systems with the capability required to make everything work together. Of course, it's not just writing the software, it's writing the software right, so that it is robust, of high integrity and self-correcting.[3]

Moore's law is still producing tremendous advances in computing power every 18 months. Yet many projects still wait as long as 6-12 months for up front project specifications, planning, and architecture to be complete. By the time developers start creating the software, the marketplace may have bypassed the project.

What is needed are some smart new rules and a fresh approach to transform the software art into a science. As it turns out, the best way to get developers to give you what you want is to set them free!

The Promise of Extreme Programming

By not adequately addressing the uncertainties inherent in the business and development environments, the established procedures the software development industry uses to guide software development efforts often don't work. Rigid methodologies generally have one thing in common: too much of the project is planned in the uncertain environment at the project inception.

What if you had a process that allowed all participants to learn and to adjust the roadmap and the destination based on that learning? With "Internet Time" driving time-to-market expectations and frenzied activity in our economies driving features and technologies, a project methodology that can adjust seems like just the ticket.

Recently, a very promising approach, created in the automotive industry in the mid 1990's, has come to the forefront: Extreme Programming.

Extreme Programming (XP) is a software development methodology that represents a throwback to the very earliest and purest days of software development when technicians ran the show. Then, software engineers enjoyed greater autonomy largely because there were few in the management ranks who understood the task at hand. (Many will tell you that hasn't changed.) What has changed is that Extreme

© 2001 Geneer Corporation

Programmers do not abdicate their responsibility to perform the software development management function. Closest to the front lines, Extreme Programmers are well positioned to deal with project slippage, feature creep, budget overruns, and risk management in general.

XP does a good job of minimizing risk in the traditional project variables – Time, Cost, Scope, and Quality. XP is a software development methodology, and it mostly concerns itself with traditional components of software. But software is developed within a project framework. What's needed is both a software development methodology and an overarching project management methodology that enables you to manage all aspects of project risk in the same way XP manages Time, Cost, Scope, and Quality.

Overcoming Project Risks

Getting your software system application on time and on budget with a robust feature set and superior quality is all about managing risk. The more novel or unfamiliar the tasks, the riskier the project. Business drivers will always demand that it be done better, faster, cheaper. But developers can't simply embark on a project with the vague hope that it will eventually deliver business value. Projects need to be:

1. Envisioned
2. Estimated
3. Planned
4. Scheduled
5. Funded
6. Initiated
7. Executed
8. Managed
9. Completed

Most process and methodology improvement efforts focus on improving the predictability of the last three activities. That's just working on part of the problem. To really have an impact on project risk, a methodology needs to address all these aspects of the project environment. The larger the project, the more critical it is to manage all these areas of risk.

Over the years, developers have realized that projects that are smaller in scope, shorter in duration, and more crisp and specific in their mission tend to be more successful than large, complex projects. Thus, one way to make projects more successful is to make them smaller and more manageable. This, in turn, makes them more responsive to market needs and lower in risk.

By combining these concepts – smaller project segments and the need to manage the whole project environment – with the best XP and other methodologies had to offer, Geneer created a new best-of-breed proprietary project management methodology called Code Science.

© 2001 Geneer Corporation.

GENEER'S CODE SCIENCE® — *BETTER SOFTWARE SOONER*

Although Code Science was originally intended to improve on the XP software development model, it ultimately resulted in a broad remodeling of software development and project management practices to deliver functional software faster. At a high level, Code Science is comprised of:

- **Service Units** – Discrete "blocks" of Geneer professional services with inputs, deliverables and resources unique to each one.
- **Segments** – 2 to 4 week time intervals representing an iteration of product development. Large problems are broken into smaller pieces and incremental results are delivered at the end of each segment.
- **Flexible Process** – Flexibility is one of the major attributes of Code Science. A high-level Project Plan and Segment Plans are created just in time. This allows the project team to quickly and efficiently respond to the inevitable development challenges and to changes in the customer's understanding of the business need and system requirements.
- **Visible Deliverables** – After every Segment, there is something to see, touch, and test. There is software that does useful work just weeks after project inception rather than at the end of the project. This helps keep the project on track, on target, and on budget.

Managing All the Variables for a Successful Project

Like all good project methodologies, Code Science seeks to ensure that the project lives within the four key variables:

- **Time** – Code Science speeds time to market in several ways:
  - **Buy time** by moving critical functionality earlier in the construction cycle.
  - **Meter time** by designing short project segments, with small discrete goals so that timeline drift can be detected and remedied early. The project team takes small bites and adjusts their plan of attack often to incorporate enhancements based on lessons learned and problems solved.
  - **Fast feedback** loops involve the customer in evaluating progress by having the application always available. This technique can save 40 percent to 70 percent of the development time over traditional methodologies.

© 2001 Geneer Corporation

- **Cost** – Code Science achieves average cost savings of 50 percent to 70 percent by following these principals:
  - **No irrelevant design** – Don't design and build "scaffolding" for future features that may never be implemented (or, with input and feedback from real users in early versions, may wind up being implemented differently than envisioned at the outset.) This is also known as "do the simplest thing that could possibly work."
  - **Executable documentation** – Save cost by doing as little paper documentation as possible. Paper documentation can become a drag on cost and responsiveness since the more you generate early on, the more you have to maintain as you go along. Excessive documentation also adds to cost by creating confusion and uncertainty if it is not updated. In Code Science, the focus is on "executable documentation." In other words, documentation becomes 100 percent unit tests. Developers learn from requirements gathering how the software needs to work, and then document it in the form of a unit test that can be executed to assure the requirement is being met.
  - **Spike solutions** – A spike solution is a very simple program that only addresses the problem under examination and which is built to explore potential solutions. Most spikes are not good enough to keep, but are used to prove-in software design ideas before building-out a design based on that software design premise. This is done in managed, contained, time-boxed efforts.
- **Scope** – Rather than attempting to enforce ever more rigid mechanisms for avoiding scope changes, Code Science embraces the opposite extreme: Scope changes are expected, welcomed, and, most importantly, planned. The process involves the customer in a recurring evaluation of the accuracy and relevance of the current scope. Orderly and controlled adjustment of the scope target is always necessary if it is no longer adequate to meet the needs of the marketplace.
- **Quality** – Delivered software functionality is only of business value when it is reliable, and of sufficient quality. To achieve quality, extensive Software Testing is "baked in" to the development process. Since poor quality leads to cost and schedule over-runs and scope under-runs, this is good for business in the long run and for the project in the short run. Rather than saving testing until the end of development, when developers are reluctant to make changes for fear of instability, Code Science mandates continuous robust regression testing from the start of development. Developers design the tests in parallel with developing the code, sometimes modifying a design to enable better testing.

© 2001 Geneer Corporation.

As a full-featured project management system, Code Science manages several other project environment variables in addition to these traditional variables, including:

- **Issues and Risks** – By segmenting projects into 2 to 4 week iterations, Code Science reduces the amount of time for the appearance of issues and risks. Often in a long project cycle, developers have to wait for things to come together, and, when they do, problems such as system incompatibilities often arise. With Code Science, there is no waiting. The work is in front of the customer sooner allowing the opportunity for correction to happen early and often. The development team moves faster, enabling improved team communication. Using speed and enhanced communication, issues and risks have fewer places to hide.
- **Resources** – The most important project resource is people. Code Science is a methodology that energizes and empowers people, and talented developers seem to prefer it to other methodologies. Further, Code Science gives great visibility into individual developers' performance and does not normalize mediocrity. Rather, it allows people to really perform up to their potential. It recognizes the superstars and the under-performers and allows developers to quickly form the very best team. Just as with issues and risks, there is no place for resource problems to hide.
- **Interfaces** – A typical problem in software development is coordinating the work and the communication between multiple parties. Project team members often need to interface not only with the customer but also with the customer's IT department and/or other third party developers to make the system work. Managing these interfaces is critical to a rapid methodology like Code Science. Because you don't want anything to slow you down, integration points need to be identified, documented, and involved in the process to ensure a common understanding. A simulator that mimics a third party's interface may have to be built simply because you're likely to move much faster than they will.
- **Change** – Change always happens. Due to increased understanding or awareness of the true business problem or a changing marketplace, a project may be half done before the customer realizes their initial product vision needs to change. Code Science makes change easier to deal with because the architecture is kept simple. You are not designing for all the "what if" scenarios — but rather only for today's requirements. By having short iterations, there are more opportunities to deal with change.
- **Delivery** – Using Code Science, you get very good at delivering software. Since there is a deliverable at the end of each 2 to 4 week iteration, it becomes routine to turn over software to a customer. Plus, since the customer has already been asked to prioritize the scope and develop the most important functionality first, the most valuable pieces of the puzzle ends up being delivered first.

© 2001 Geneer Corporation

The Principles of Code Science

At the heart of Code Science are practices and principles that, when combined in this unique way, comprise just the right amount of freedom and discipline to get things right. Kent Beck[4], the founder of Extreme Programming, sheds some light on what really matters in software development:

1. *Coding.* At the end of the day, if the program doesn't run and make money for the client, you haven't done anything.
2. *Testing.* You have to know when you're done. The tests tell you this. If you're smart, you'll write them first so you'll know the instant you're done. Otherwise, you're stuck thinking you maybe might be done, but knowing you're probably not, but you're not sure how close you are.
3. *Listening.* You have to learn what the problem is in the first place, and then you have to learn what numbers to put in the tests. You probably won't know this yourself, so you have to get good at listening to clients – users, managers, and business people.
4. *Designing.* You have to take what your program tells you about how it wants to be structured and feed it back into the program. Otherwise, you'll sink under the weight of your own guesses.

In other words, everything that doesn't help a developer code, test, listen and design needs to be pruned away from the project management methodology. One of the most important parts of the developer's job is to listen. One way Code Science helps this communication is through something called User Stories.

User Stories Drive Results

User Stories are informal descriptions captured in the customer's words about how users will use the system. Customers should be encouraged to make up personalities and names for the different types of system users. This makes it easier to relate to their points of view. It may seem a little silly at first, but it helps to put a name (and a face, depending how imaginative you get) to the users of the system. All customers need are some index cards, a pencil, and a little time. The stories you're after are those that describe the users of the system performing their respective tasks.

Let's say you are writing an online ordering system. User personalities may be Carl (the customer), Melanie (the manager at the warehouse), and Sam (the stock boy). By putting names to the users, you can have the end "customer" in mind at all times. You'll often find development team members saying things like, "Carl would find this confusing," or "Melanie would think this feature is a must-have."

© 2001 Geneer Corporation.

For example, consider the following User Stories:

1. Carl logs in to the system
2. Carl searches the catalog for widgets
3. Carl orders widgets

In a quick and simple way, the customer has indicated:

1. The system requires a log in, which implies some sort of security system
2. There is a searchable catalog of widgets, which implies a database and a search methodology
3. Users can initiate order transactions, which implies a transaction execution, monitoring, and roll back system

Obviously, these various system components need to be further fleshed out, and the customer may be asked to create more stories to handle various uses of the system. User Stories are the basis for managing scope and planning construction activities. They represent distinct pieces of the system that are usable and of value if delivered on their own.

To help your customers identify and understand User Stories, you can provide guidelines and some examples. If customers feel uneasy about being locked into brief specifications, your motto should be, "We want to deliver what you need, not necessarily what you have asked for in writing."

Small Teams, Small Iterations are Efficient

Teams of ten or fewer tend to better focus on fulfilling the customer needs. Code Science relies on tight communication across team members. One way to achieve this is to have the entire team take part in *Iteration Planning*, a process that designs each small, 2 to 4 week incremental effort to deliver User Stories. To foster communication, all team members attend a short, daily *Stand Up* meeting. It gets its name from the fact that, almost universally, they are held in the hallway, in front of a Code Science innovation, the Wall Gantt™, which is described in the next section.

The best teams get their Daily Standup meetings done in 15-20 minutes. Team norms dictate that they begin on time and that it is disrespectful to your teammates to be late. Since team members stand instead of comfortably lounging in chairs, there are certain incentives to ensure meetings are short and to the point.

© 2001 Geneer Corporation

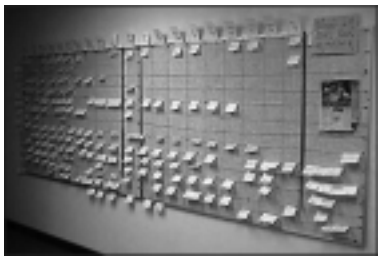
Wall Gantt™ Manages Status and Accountability

Originated by Geneer, the Wall Gantt™ is a large task schedule posted on the wall, where the entire team will see it frequently. This process clearly and simply communicates the plan for the near future.

Teams find that the Wall Gantt has the following benefits:

- **Focal point** for Stand-Up meeting discussions – Tangible progress is separated from vague rationalizations as tasks are marked complete or rescheduled on the wall with the help of the team.
- **Simple and accessible** - You don't have to be a whiz at MS Project to see where things stand, or to update your own progress. You can also see if there are others needing help, and where you can pitch in.
- **Scale constrains scope** - If you can't manage a team's activities with a Wall Gantt then the a) the scope's too big, b) the team's too big, or c) both.
- **Provides clear ownership** of development efforts – Each task is labeled with the owner.
- **Encourages accountability** for assigned work – During a Stand-Up meeting, team members can question one another about progress and roadblocks.
- **Provides visibility** into current progress – Anybody walking by, including customers, can see at a glance where the project is.

The earliest incarnations of Wall Gantts were done with dry erase markers on whiteboards. While this afforded visibility, it was difficult to maintain. Also, the size of whiteboards constrained the communication of a sufficient amount of task-level detail. Teams migrated to cards, and progress monitoring and morale management needs led to the introduction of stickers. Happy face stickers of encouragement have become common on some teams to indicate completed tasks. Further evolution resulted in the use of taut strings to line up tasks.



© 2001 Geneer Corporation.

Buddy System – No One Codes Alone

While team members have their own offices with doors, it is sometimes helpful to have a more senior person sit with a newer or less experienced person while they work to decrease ramp-up time. Another Code Science technique is Pair Programming, the assignment of two programmers to the same task, usually done to bring in a critical path task sooner. Though you don't often see an immediate 2X gain in productivity, with a critical path task, you can be pleased with anything greater than 1X.

For any component of the system, or for any critical responsibility there is always an identifiable Primary and Secondary resource capable of working on it. Despite this, Code Science encourages single ownership of defect resolution, so that a person working on defect does not hand it off when he or she traces the cause to someone else's code. Rather, they prepare a repair and review it with the other developer. In any case, in Code Science, program code belongs to the project, not to an individual engineer. Additionally, one or more peers inspect all work to find defects early and also spread knowledge of the work product.

Do The Simplest Thing That Could Possibly Work

This concept is central to the Code Science approach. Program functions should be as simple as possible, and no simpler. When redundant functions turn up in the course of a project, teams right size them using a process called Refactoring. Refactoring can be thought of as distilling the essence of the software to discover the simplest way to satisfy the system requirements.

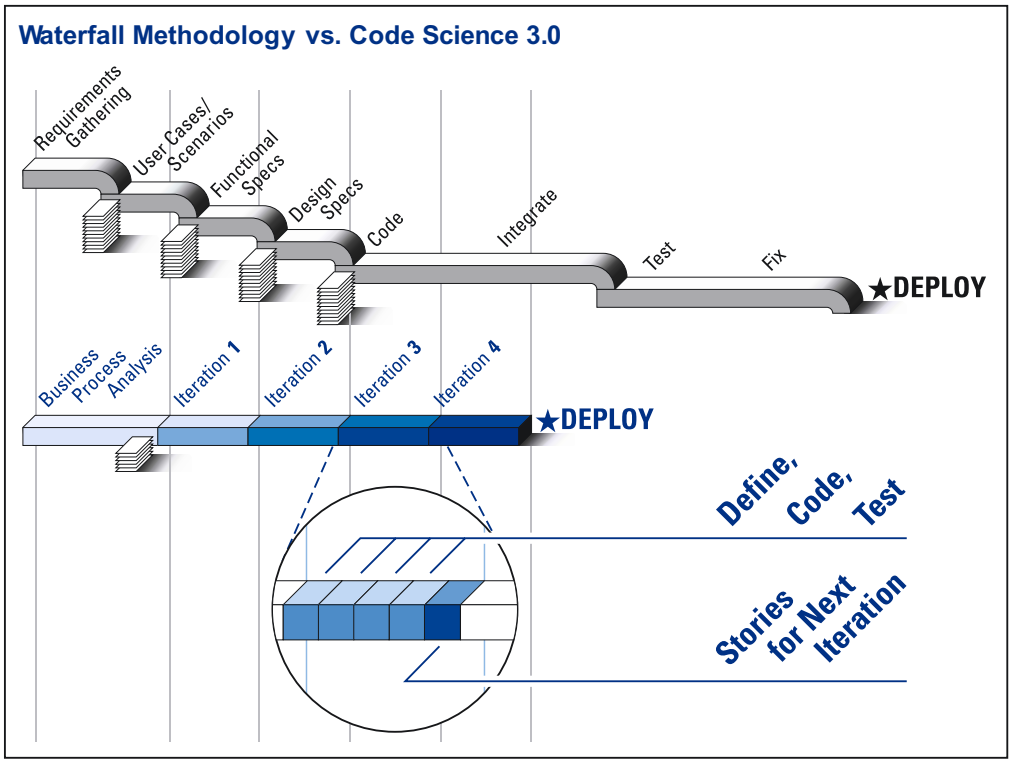
Test Early and Often

As we described in a previous section, the Code Science approach disdains extensive paper documentation in favor of Unit Tests that define code functionality. Unit testing in Code Science involves developing and running automated tests on code to find defects early. Unit tests for code are written before the code itself. We write automated tests for each method, run them quickly and often, and keep those tests forever. Code must pass all tests at all times.

© 2001 Geneer Corporation

It's Not a Waterfall, It's a Productivity Engine

Perhaps the most popular form of project management methodology is the Waterfall method. In the diagram below, the Waterfall is represented by the gray figures. First you gather requirements, and then you do use cases, which are similar to User Stories. Next you do your up-front planning, the functional specs and the design specs. At last you start to code, then you integrate, test, and fix bugs. Finally you deploy. The little stacks of paper in the diagram indicate the copious documentation produced in a typical Waterfall project.



Code Science, as you've gathered, does things differently. The up-front planning consists of User Stories and communication with the customer to understand the project objectives. During this phase, the customer will prioritize User Stories so that the important functionality can be delivered first. The team does some Iteration Planning, and embarks on the first iteration, generally 2 to 4 weeks of effort. The first iteration results in usable code that the customer can try out to see if it satisfies the User Story. If the software misses the target, all parties adjust and ensure that it does. The User Stories for the next phase are either generated or revisited, the next iteration is defined, and you're off and running again. Each iteration follows in turn, with unit testing representing a combination of status reporting and quality assurance. Then, usually weeks or months before a comparable Waterfall project, the software is deployed.

© 2001 Geneer Corporation.

HOW YOU CAN BENEFIT FROM USING CODE SCIENCE

Code Science enjoys many advantages over traditional project management methodologies. In addition to delivering software better, faster, more cost effectively, and with higher quality, Code Science improves project management, customer/team communication, and system needs discovery.

Eliminates The Need For "Herding Cats"

Even though Code Science is a methodology with heavy emphasis on making developers' jobs easier, when properly applied, managers enjoy substantial benefits as well. First of all, Code Science is a participatory project management methodology. This means there is less need for a centralized project management responsibility as in traditional methodologies. The traditional project manager, responsible for resource assignment and leveling, task assignments, time collection and status reporting, morphs into a shared, and in some cases invisibly satisfied, responsibility of the team as a whole. In other words, developers manage themselves. Developers meet daily to discuss their progress on small, discrete tasks, and can determine how to best adjust their workloads to maintain the desired project velocity.

Determining and communicating project status is also a self-solving problem, because the software serves as the status report. Daily builds and continuous review of new features by the customer provides a constantly updated and collectively shared understanding of where things stand.

Therefore, managing a Code Science project, rather than being akin to "herding cats," is more about providing the necessary resources, giving feedback on the progress of features, and staying out of the way.

Make Software Projects Understandable

Among the frustrations that customers of software development efforts have historically faced is the difficulty of grasping just how their requests are being interpreted and transferred into working software. Software developers have a tendency to speak in terms that are obtuse or meaningless to the average end user. Code Science demystifies the art of software development through the central positioning of User Stories in the project. Since User Stories are written by the customer, in the customer's own words, and all evaluation of project outputs is done using User Stories, there is a clear common language between customer and developer.

© 2001 Geneer Corporation

Discover New and Evolving Needs

Code Science projects enable organizations to gain or maintain competitive advantage because of its swiftness and acceptance of change. In addition, Code Science exposes latent customer needs through fast feedback. By continuously presenting the customer with working software to address the User Stories, the team can learn what the customer really wants, which can be different from what he or she initially described. Thus, development can be married with business strategy, even as it shifts over time. The organization can try more options due to the decreased investment required up front and the fast cycle time. Thus, you can increase project throughput, getting more projects done in less time.

Experience High Quality

Code Science projects attain a high level of quality through use of coding standards, high customer involvement that ensures the proper direction as well as frequent Unit and Acceptance Testing that ensure the system works and satisfies the business needs. Code Science gives the development team the freedom and the responsibility to ensure that their work meets the highest quality standards. Performing constant testing throughout the development process rather than in a rush at the end allows the team to make adjustments along the way to avoid cascading mistakes throughout the system.

Improve Project Speed

Code Science delivers unparalleled project speed. This means you will be able to pursue more opportunities in the software you deliver, since you're not struggling just to meet the dates. And the sooner you get the goods, the sooner you can turn expenses into revenue. Code Science achieves its speed in many ways, but a very important one is its ability to solve critical issues early through the use of User Stories, incessant Unit Testing, and swift iterations. This methodology is thus optimized for responding to market changes. Now you can get to market before the next technology wave leaves you in its wake.

Improve Return on Investment

Because the most important business functionality is built first, mistakes are caught early, ensuring that the project's direction is correct and resulting in rapid, significant ROI. Additionally, by developing in tight iterations, you can quickly determine if value for functionality is still positive. If the answer is no, you can either work with the team to adjust the targets, or even abandon the effort and begin anew, incorporating new technologies and learning if the market has passed you by.

© 2001 Geneer Corporation.

CONCLUSION... AND A FEW WORDS OF CAUTION

After reading this, it must seem that we're claiming that Code Science is the best thing since sliced bread. Well, it's awfully good, but it's not appropriate for every potential project. Code Science is not the preferred methodology when strict regulatory issues are to be addressed, or there are safety issues at hand. Generally such projects require a higher level of documentation and process than Code Science typically delivers. Code Science is all about unleashing the creativity of the team, and saddling them with old-style reporting procedures is a disservice to the methodology. You're better off with a traditional development process (which, incidentally, we can still do at Geneer).

Code Science does not work if customer demands cause iteration lengths to stretch. It does require a certain amount of trust to abandon the old ways of doing things, and customers that can't get with the Code Science flow are better off using methodologies they are more accustomed to. Code Science is not entirely immune to customer demands that affect the process by demanding, for example, that testing be abbreviated or scope unreasonably stretched.

Basically, as a customer with your first Code Science project, you will need to trust the team a fair amount until you get the hang of it. If the approach seems totally crazy to you, if you think it's insane not to have business requirements nailed down before proceeding, or if you have trust issues, perhaps Code Science is not for you. But if you can get past a few of the early hurdles to see some results, chances are you'll be able to go with the Code Science flow.

One thing that anyone contemplating a dip in the waters of Code Science must recognize is that you will not get it entirely right the first time. In our experience, employing Code Science on many projects, each project team and each customer must be prepared to go through a period of adoption. Although many Code Science practices seem like simple, common sense advice, some of them are much more difficult to implement than others.

As you can see, Code Science has only a little to do with programming and has a lot more to do with project execution and management. It challenges some closely held beliefs about the "right way" to develop software. So, know this, and go with the flow. The Code Science tide appears to be taking us where we want to go: *Better Software Sooner*SM. And if it doesn't seem to be getting you where you wanted to go, you can always change your tack and keep trying.

© 2001 Geneer Corporation

APPENDIX A: EXTREME PROGRAMMING RESOURCES

Many of the concepts on which Code Science is based come from Extreme Programming. Here are some links to resources if you want to learn more. Beware, however, these are all more suited to the highly technical programmer than to the business-oriented manager. Keep in mind that Code Science adds significantly to the XP development methodology to offer a full-featured project management methodology.

Extreme Programming Roadmap, on Ward Cunningham's Wiki
http://c2.com/cgi/wiki?ExtremeProgrammingRoadmap

ExtremeProgramming.org
http://www.extremeprogramming.org/

XP Developer
http://www.xpdeveloper.com/

Ron Jeffrie's Xprogramming.com
http://www.xprogramming.com/

William Wake's XPlorations
http://users.vnet.net/wwake/
Iterative development resource:
Book: "Rapid Development" by Steve McConnell

Extreme Programming:
Book: "Extreme Programming Explained: Embrace Change" by Kent Beck

User stories and personae:
Book: "The Inmates Are Running the Asylum: Why High Tech Products Drive Us Crazy and How To Restore The Sanity" by Alan Cooper

BPA, feedback and high client involvement:
Book: "Joint Application Development" by Jane Wood, Denise Silver

Frequent testing, programming by contract:
Book: "Object-Oriented Software Construction" by Bertrand Meyer

FOOTNOTES

1 http://www.vbpmj.com/articles/whyprojectfail.htm

2 ibid.

3 "The Evolution of Computing," Bruce Powel Douglass, Software Development, January 2001

4 http://c2.com/cgi/wiki?ExtremeProgramming

Geneer and Code Science are registered trademarks of Geneer Corporation. All rights reserved.
Geneer clients may make one attributed copy or slide of each table or graphic contained herein.
Additional reproduction is strictly prohibited. For additional reproduction rights and usage information, please contact Geneer at 800-4-Geneer. Information is based on best available resources.

© 2001 Geneer Corporation.

ABOUT GENEER

Established in 1984, Geneer is a leading professional services firm specializing in the design and development of custom software applications. With a strong track record in using emerging software technologies to make its clients competitive in their market place, Geneer's longevity, dependability, and thorough understanding of technology and software development have earned it the honor of working for an outstanding group of clients, including many of the Fortune 100. Geneer was recently named as the first Microsoft Gold Certified Partner for eCommerce Solutions in the U.S.

GENEER EXECUTIVE BRIEFING SERIES

Geneer's Executive Briefing Series delivers essential insights on the future of e-business, the Internet, and the technology that can have a significant impact on your product and service offerings. If you are interested in receiving future Geneer white papers, please contact Geneer Business Development: 800-4-Geneer or 847-294-0300. For more information about Geneer, visit our web site at www.geneer.com or e-mail us at info@geneer.com.

© 2001 Geneer Corporation

Building Software. Building Business.℠

Geneer®

1400 East Touhy Avenue, Des Plaines, IL 60018-3340
phone: 847.294.0300 • toll free: 800.443.6337 • fax: 847.294.0358 • www.geneer.com