
METHODS & TOOLS

Global knowledge source for software development professionals

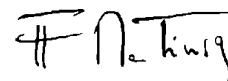
ISSN 1023-4918

Summer 2002 (Volume 10 - number 2)

You Need Four Eyes

Software development is considered a very individual task and most programmers think that they do a good job and that they produce good programs. As we all know, this not exactly true. Most of the time, there is still a lot of work to do between the time when the program is "finished" and when it works well enough for the user. There are two different approaches to remove defects from programs. The mostly used, is a reactive one: "the program is finished, let's test it to see if it works". The other one is proactive: "the program (or design if it exists) is finished, let me see if it is a good one". Software inspections or reviews are an old concept that is based on the fact that four eyes (or more) are better than two. Even if similar ideas are adopted for instance by the eXtreme Programming approach ("pair programming"), software inspections have proved difficult to implement. Adoption has to deal with the developer's ego and its introverted personality. To avoid personal conflicts, programs are considered as "personal black boxes" in many organisations, until the programmer leaves and somebody else has to maintain it. Then, they became simply "black boxes"... or sometimes "black holes".

To implement software inspections, both the developer and the reviewer need to make efforts. The developer should accept to show its code. He must be willing to learn from others and have time to do it ("with those deadlines, are you kidding?"). The reviewer has to accept other solutions. He should know how to suggest without offending and how to cope with the "judge" role ("I am here to develop, not to take care of someone else's code!"). There are however many advantages in having a software inspection program in place. The most obvious is having fewer errors in the code. Not only can the reviewer find some of them, but the fact that the developer need to explain how and why he did it this way will give positive result. How many times did you go to a colleague looking for help and found the solution yourself while you were explaining the problem? Other possible benefits are: more readable code (because it must go through inspection), emulation between developers, backup of system knowledge as two developers will understand what a part of it should do and sharing of software development ideas as good developers recognise clever code when they see it in a program. It is not easy to put in place an inspection program, because it touches the "intimacy" of programmers, but I think the results are certainly worth the try.



Inside

Strategy: Observations on Strategy Used in IT Organizations Today..... page 2

Quality: Software Inspections page 7

Observations on Strategy Used in IT Organizations Today

Jason Charvart, info@chasoncharvat.com
www.jasoncharvat.com

Sometimes all this talk of business strategy, competitive edge, and technology gets a little hard to digest all at once. In the course of my work as a project consultant, I notice on a daily basis how rapidly computer software and technologies change, and it's getting difficult to keep up. Before you know it, another version of software is being introduced or a newer technology is on the market. Today, you can get state-of-the-art software applications that can be developed far more quickly than before, allowing organizations improved functionality and opportunities.

Senior executives face the front line with a constant bombardment of software companies and consultants who market IT solutions, which are able to revolutionize and improve their organizations. Sadly, not many of these software systems get developed or implemented correctly as what the client would have liked it. The most important predictor of an organizations ultimate success or failure is the strategy that it chooses to adopt.

To build effective products or services, you'll need to focus on the hard stuff. Be it Project managers, methodology, tools, supportive processes or even deploying a Project Management Office (PMO) one needs to understand how strategy plays such an important part within their respective organization. Remember: You can't build a competitive project-centric culture by simply presenting a "Strategy" session to your executive team and expect you'll have a Sun-Tzu like organization in place overnight. Building a professional project organization takes time – and great strategy.

Some of the observations I have found in Fortune 1000 organizations are:

- Projects are simply assigned to individual project managers to complete, irrespective if they suit the overall strategy.
- There is no coordination or interface among individual managers as to what the strategy is
- Project managers are assigned projects too late and one can never catch your competition
- Minimal project processes are defined based upon the newly chosen business strategy
- Communicating the organizational strategy downwards is rarely done and many managers are unable to accurately translate what the executive's vision is.
- There is ineffective adherence to measuring whether or not one is achieving the set strategy
- The information technology is sometimes not included into the overall strategy

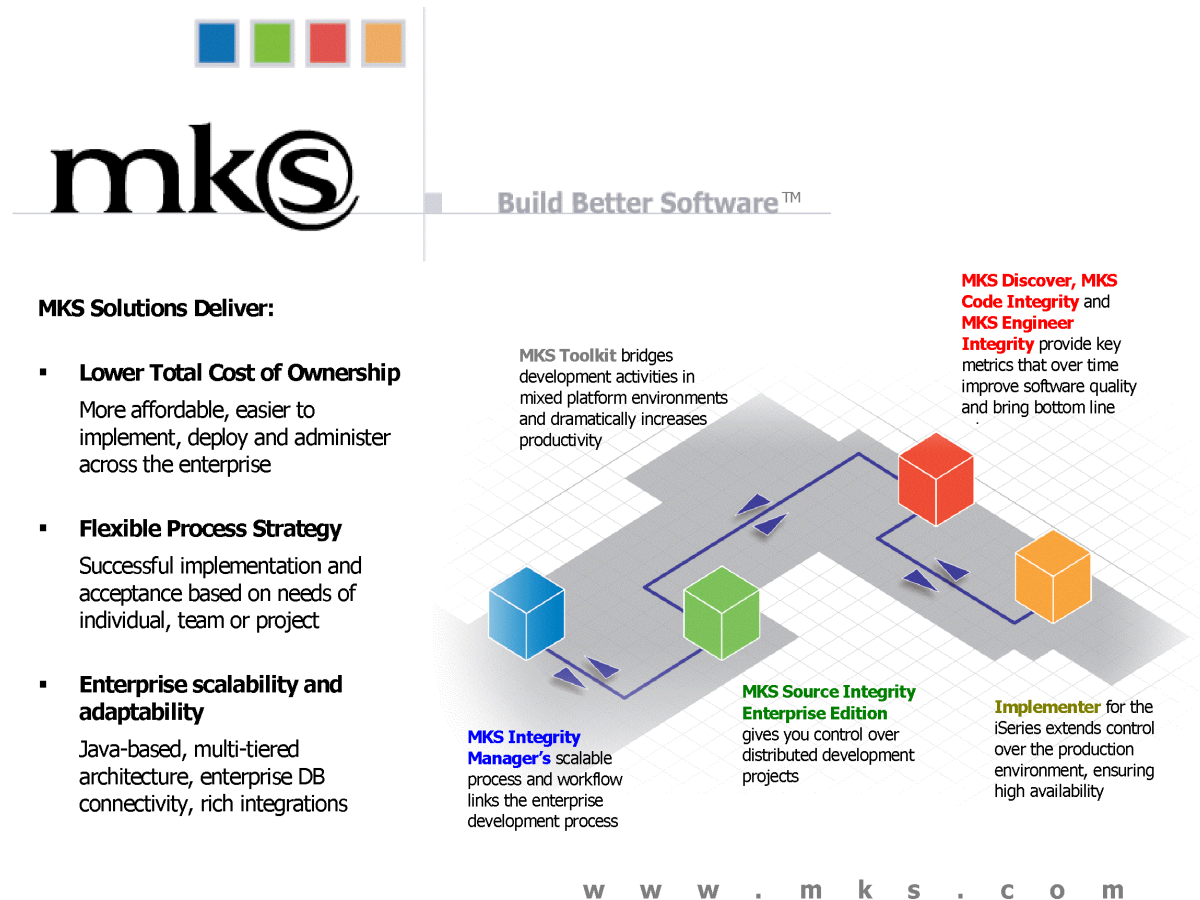
The latest Tom Clancy movie, "Sum of all Fears", provides an interesting metaphor for the challenges of actually deploying a strategy. During one scene, CIA analyst, Jack Ryan, races against time trying to persuade both America and Russia, not to enter into a nuclear war. Bu both countries have well defined nuclear strategies, and as each superpower starts preparing to deploy this strategy, things dramatically get more interesting. In a few nail-biting scenes, Ryan manages to persuade the world leaders that the threat is fallacious and incorrect, and the countdown stopped! The scene clearly shows how each country had a clear well-defined strategy, which could be implemented, when needed.

In this elegant but simple allusion, is compacted the entire issue of having a clear-cut strategy; irrespective of the question of whether you will be successful or not. So too, projects need to rely on a proper strategy in order for any successful project to come to light. Get the picture!

Organizations are challenged, as they need to keep pace with competitive markets, client needs and futuristic trends. It basically goes about who has the upper hand – either with new technology, tools, methodologies or quicker implementations - the only winners will be those executives that are able to reinvent their companies quickly enough to take full advantage of the efficiencies and distribution that new technologies can afford. To overcome your competition and to be an industry leader, you need to be able to provide your clients the latest products and services available. And project management plays an important role in all of this. The strategy must be deployable!

However, before one can get to the point, of deploying a product or solution, it requires a strategic assessment and planning before commencing. The executive team within the organization needs to “envision” a strategic plan or game plan – to use a sports metaphor – before any deployment takes place.

Without a strategic plan in place, executives will literally see their organization moving from one solution offering to the next, spending millions of dollars, which will result in many projects heading south. The point after all is to make sure the organization is more valuable, has a business strategy in place and is ready to start deploying this game plan.



The MKS logo features four colored squares (blue, green, red, orange) above the text 'mks' in a stylized font. To the right, the tagline 'Build Better Software™' is displayed. Below the logo, a diagram illustrates the integration of various MKS tools. The diagram shows a central path with four colored cubes (blue, green, red, orange) connected by arrows, representing a workflow. Each cube is associated with a specific tool description:

- MKS Integrity Manager's** scalable process and workflow links the enterprise development process
- MKS Source Integrity Enterprise Edition** gives you control over distributed development projects
- MKS Discover, MKS Code Integrity and MKS Engineer Integrity** provide key metrics that over time improve software quality and bring bottom line
- Implementer** for the iSeries extends control over the production environment, ensuring high availability

MKS Solutions Deliver:

- **Lower Total Cost of Ownership**
More affordable, easier to implement, deploy and administer across the enterprise
- **Flexible Process Strategy**
Successful implementation and acceptance based on needs of individual, team or project
- **Enterprise scalability and adaptability**
Java-based, multi-tiered architecture, enterprise DB connectivity, rich integrations

w w w . m k s . c o m

From a project management point of view, there's no point in managing any project if you have no idea why you're doing it in the first place. It's crucial for any project manager to address the larger issues of the business strategy and see where the project fits in the overall framework. It isn't easy – but it needs to be done. The thoughts contained within this article are important as they represent the strategic concepts and ideas formulated at the corporate or business level and the role of the project manager at a lower functional or operating level. When we talk about business strategy, I am also including the alignment of information technology as an integral part of the game plan. The reason may be that companies are reluctant to invest in new technologies may therefore never address their IT problems or worse are left behind by their competitors. Therefore organizations need a documented strategy that is realistic and one that everyone agrees to. Good strategy leads to good results. Bad strategy will not allow an organization to survive its competitors.

Let me illustrate an example of how technology and market trends are forcing organizations to adapt their business strategies to meet future information technology demands. It is estimated that by 2005 over 80 million people will be sending wireless images on the fly, using numerous digital devices. Sounds like something from Star Wars? Doesn't it. If this is the case and it happens, existing network infrastructures will be outdated and placed in risk, as newer high-speed networks on the 128 kbps and 384K TDMA range will be needed to handle these technologies. The strategies of many companies will therefore need to be revised at both the business and IT levels, and the actions that lead out of these strategies will be required to be implemented by project managers.

It is therefore clear that if your department runs multiple projects in support of the strategy, then your organization must master project management as a method in order to be successful in the marketplace. A PMO is another key business strategy used by companies to manage their projects, in order to maximize their value to the organization. It's almost a mind-set, a way that reshapes entire organizational processes. As the industry faces change, so must their departments and the PMO simply adjusts the project portfolio accordingly. If not, then they're missing key opportunities to provide value to their clients.

Departments running projects should perhaps be taking a new look at how a PMO can maximize and contribute towards getting its products and services - designed and deployed - quicker to market. At best, a new approach may awaken old processes, which currently prevent you from moving faster than your competitors. For those IT groups that have major problems getting projects initiated, prioritized or brought under control, the utilization of a PMO is the first step in the right direction.

The first and most important fact is that the business strategy be developed and be set in motion for the organization. The IT strategy then forms the core part of how to get there. These strategies must be verified and discussed at an executive level, where IT is involved. If the overall strategy is wrong or the problem strategically misunderstood, it is not surprising that the results are less than satisfactory. No amount of effort or leadership or tactical brilliance from your executive leaders will compensate for an incorrect strategy. Strategies are always formed and executed at different levels within any organization.

Information technology is changing at such an amazing rate that more and more solutions, require enhancements to existing systems, and de-commissioning of older systems are required in order for companies to survive in the competitive marketplace. So, too, project management needs to fit into the overall company strategic model, whereby project management is the area that brings in the IT solutions (products or services) before competitors can react.

By applying project management, and having an understanding of the strategic intent of the company, it becomes all the more important to maneuver the advantage correctly. Projects need to bring in solutions faster, cheaper or with a cost advantage, which is unique, focused, and be able to serve clients worldwide. Sun Tzu, a famous military general once said:

“The one with many strategic factors on his side wins.... The one with few strategic factors on his side loses. In this way, I can tell who will win and who will lose.”

The project manager has to take the slog up the mountain and ask tough questions from the project sponsor and other stakeholders: “How do we measure success at the end of this project?”. “What do you really want to buy for all this money we're going to spend?” To get answers to these questions, everyone must examine the strategic aspect, which starts at the very beginning of the project idea or concept. Without an understanding of the desired result, the project manager cannot fend off scope creep and define success for the people who will be doing the work.

Across many industries – especially in IT, projects encounter many of the same old problems time and time again, irrespective of their geographic locations. I have heard project managers in the UK, Belgium, India and even Puerto Rico complain bitterly about similar issues on IT projects. The strange thing is that many were utilizing different project methodologies; templates, reporting tools and communication seemed to be a big problem affecting all.

The purpose of strategy is to provide direction and concentration of effort as organizations continually strive to improve their position or gain the upper hand within the marketplace. Basically, it's a struggle for advantage, and the one with the best advantage wins. It's that simple.

What must businesses concentrate on? Businesses clearly have to:

- Achieve new advantages that increase or improve customer satisfaction, which differentiate them from their competitors.
- Either eliminate or minimise their competitors.
- Achieve speed to market.
- Re-engineer business processes for improved competitiveness.
- Align their organizations to the latest economic trends.
- Implement the strategy (i.e. through projects).
- Evaluate the success of the strategy (i.e. measure project success).

IT departments are increasingly looking to a strategy to provide solutions to many of the challenges and they need to react quickly. Projects need to be managed throughout their life cycle. IT departments manage on average up to ninety (90) projects each year, each varying in size and complexity, and considering only 16.2 %¹ achieve success, it drives one only closer to the fact that the strategy must be focused.

IT departments therefore need to:

1. Re-examine their current project management practice.
2. Determine current strategies.

¹ The Standish Group International – The CHAOS report

3. Identify bottlenecks.
4. Be able to act on existing projects.

It is worth mentioning that my view is framed within the idea that companies must adapt to accommodate and serve the competitive business models of the future. So if we can challenge existing IT project practices review their advantages and disadvantages one would be able to improve. Its obvious and glaring that an organization, which is “projectized” and has an innovative strategy in place will pave the way for the future of many organizations. Just look at innovative companies such as Nokia, Disney, J&J, Virgin, Honda and Novartis. They have all overcome project obstacles, by introducing and implementing excellent strategies.

In conclusion, IT projects need to be managed centrally. What is the secret to any organizational success? By providing solid, repeatable strategies, which serve as the foundation for any successful project initiative, supported by

1. Well-documented goals and visions.
2. Repeatable best practices.
3. Consistent results.

If IT departments can achieve this and implement the strategy and manage all their projects on time and cost, then maybe, just maybe, one can be confident that this will allow projects to be designed and be deployed quicker than before, thus enabling them to become a truly project-centric organization.

© Copyright Jason Charvat

Software Inspections

Ron Radice, rradice@stt.com
Software Technology Transition, www.stt.com

When an old idea is a good idea that improves to become a better idea, we should all want to benefit from that evolution. Software Inspection was a good idea when started in 1972. Inspections have continued to provide a quick return on investment and perhaps one of the quickest for all the methods and processes available to the software practitioner.

In a recent editorial in Informationweek [STA02], Stephanie Stahl, the editor, says “In order to have great software, companies have to make completion dates and deadlines a matter of history.” She then quotes a Steven Jones who gave her this input, “Bottom line, it will be passed to the next phase when it’s right. No dates, no expectations set. When it’s right, it’s ready.” Well, as has been known for some time, Software Inspections not only make it right, but also makes it ready at a lower cost. How can anyone in business who believes in quality resist such an offer?

The following excerpts are from my recent book, *High Quality Low Cost Software Inspections*, and are included here with the permission of my publisher. [RAD02] While I cannot in this brief article address all aspects of the Software Inspections evolution since 1972, I believe you will find excerpts below that may stimulate you to further exploit Software Inspections. And if you are one of those who have cast away Inspections for whatever reason, I sincerely anticipate that you will try them again to your advantage.

Introduction

I have never found a manager or programmer who purposely wanted to deliver a product late, over cost, or of poor quality. Yet projects continue to ship late, with less function than committed, at higher costs, and with questionable quality. There are a number of factors that lead to these undesirable project results, but a major contributor is the lack of defect removal controls. Defects are created or injected into work products throughout the project life cycle. This seems to be an unfortunate fact of software development. It can be difficult and expensive to remove the defects in test, and when customers find defects the costs can increase by a factor of 100 or more.

These costs to remove defects are part of the Cost of Quality, or more significantly the lack of quality, and can represent as much as 65% of total project costs. Clearly there is economic opportunity to improve both the quality and consequently the return on investment (ROI) for software projects.

In the book I address both subjects using two primary processes:

1. Inspections to find defects earlier and at a lower cost
2. Defect Prevention to reduce the volume of defects injected

The software community has used Inspections for almost twenty-eight years. During this timeframe Inspections have consistently added value for many software organizations. Yet for others, Inspections never succeeded as well as expected, primarily because these organizations

did not learn how to make Inspection both effective and low cost. We will see that in fact the cost of Inspections is very often paid back with a handsome return during the first project's use.

I have seen Inspections work successfully time and time again to the surprise of some of its staunchest opponents. At the same time I have seen Inspections abused, misused, and aborted for some of the most shortsighted and irrational of reasons.



**The lesson to be learned from these experiences
is that methods and tools can be misapplied, treated as a failure,
and then dismissed as a bad experience
by users who were not enabled for success.**

The Inspection method itself is simple and straightforward, but it does require a belief in its capabilities, application of necessary preconditions, and good management support to make it work to a software organization's best advantage. If management does not support the principles of good quality, then overworked programmers and managers will find innumerable excuses to cause Inspections to fail. If the software managers and software engineers in the organization do not believe that the process will work, then there is a good chance that they will fulfill their expectations.

On the other side, given a fair chance, support by management, a commitment of some early time investment in the project, proper training by example, and practice of proven principles, the Inspection process will take root and work effectively and efficiently. When practicing Inspections one should always work to achieve effectiveness first, then, while maintaining high effectiveness, work to improve the efficiency. Throughout the book we learn that this is the desired two-step with Inspections: first effectiveness, then efficiency.

Why Inspections?

So what's the problem? Why do we need Inspections? We need Inspections to remove software defects at reduced cost. Inspections enable us to remove defects early in the software life cycle, and it is always cheaper to remove defects earlier than later in the software life cycle. It's important to note that Inspections are a way to remove defects at a lower cost, not a way to prevent defects from occurring.

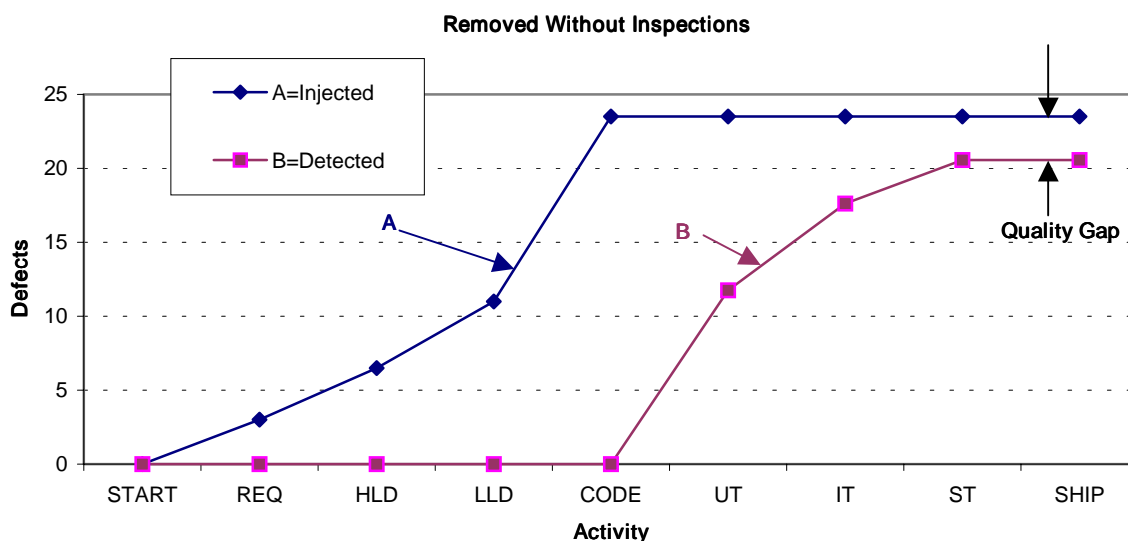


Figure 1. Example of Defect Injection and Defect Removal Curve

In Figure 1 the curve A from START to CODE represents the numbers of defects that were created or entered into the production processes during requirements analysis (REQ), High-level design (HLD), Low-level design (LLD), and Code in a software project. Curve A shows no pre-test defect removal activities. Consequently each test activity; e.g., (Unit Test (UT), Integration Test (IT), and System Test (ST)), as shown in curve B, finds and removes varying volumes of the generated or injected defects.

In this example let's assume that each test removes 50% of the defects present at entry to the test; e.g., UT, IT, ST. In a well-defined, well-behaved series of tests, we would expect that the number of defects removed would decline in each subsequent test activity. While this is a simplistic view, evidence to the contrary would suggest a quality problem for the product under test. In this example we will make the assumption that the tests are well behaved. The gap that remains at the end of system test (ST) represents latent defects that have the potential to be found by the users. Latent defects are defects remaining in the software product delivered to the customer at SHIP.

So far in this example we can see that:

- Defects are generated in each life cycle production activity
- Injected defects are removed in testing activities after code is completed
- Not all defects are removed at SHIP

Had we tried to deliver the product at the conclusion of coding, we would most probably not have had a working product due to the volume of defects. There is a cost to remove defects found during test, and as we'll soon see, the cost to remove defects during test is always higher than in pre-test. Users of software products all too frequently encounter problems and we can assume that these defects will usually require resolution and repair and therefore these are an added cost to the software project or organization. Both of these costs will vary for a number of reasons, such as the number of defects entering from each production activity, the types and severities of defects, and the quality policy in the organization. The sum of these costs increases the Cost of Quality for the project, where the Cost of Quality is money spent due to poor quality in the product.



Our objective with Inspections is to reduce the Cost of Quality by finding and removing defects earlier and at a lower cost.

While some testing will always be necessary, we can reduce the costs of test by reducing the volume of defects propagated to test. We want to use test to verify and validate functional correctness, not for defect removal and associated rework costs. We want to use test to prove the correctness of the product without the high cost of defect removal normally seen in test. Additionally we want to use test to avoid impacting the users with defective products.

So the problems that must be addressed are:

- Defects are injected when software solutions are produced
- Removal of defects in test is costly
- Users are impacted by too many software defects in delivered products

With Inspections we can manage and reduce the effect of these problems. In Figure 2 the same curve (A-Injected) is shown as in Figure 1. An additional curve (C-Detected with Inspections) represents defects remaining after removal by Inspections for the volume of defects injected and

remaining during each production activity as shown in curve A. In this example the assumption is 50% defect removal effectiveness for each Inspection and test activity. Effectiveness is the percentage of defects removed from the base of all the defects entering the defect removal activity. As is seen in Figure 2, the number of latent defects in the final product is reduced with Inspections.

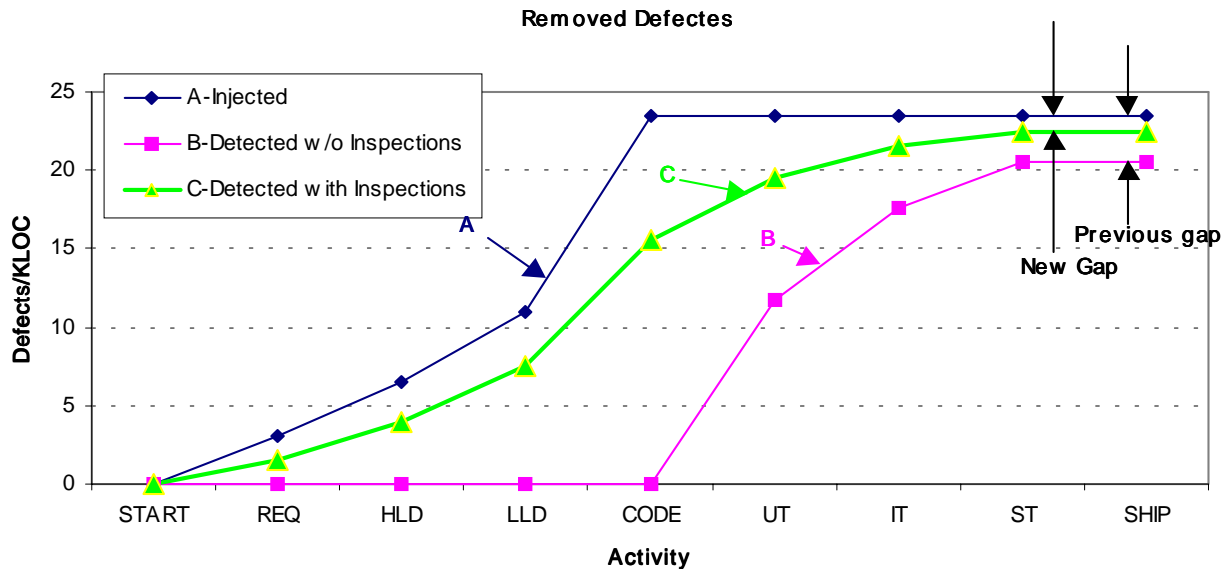


Figure 2. Example of Defect Removal With Inspections

After Inspections, residual defects are removed during testing, but typically not all injected defects are removed. There is still a gap of latent defects that the users could potentially find. In the scenario, with Inspections, the gap is smaller, due to the defects removed by Inspections. This reduced gap represents a quality improvement in the product delivered to the users.

Ok, so we have removed defects earlier, but why is it cheaper you may ask. It is cheaper primarily due to the following:

- Defects are not discovered all at once during test or by the users. There are likely to be a number of cycles of finding defects, fixing them, and integrating the fixes into each delivery to test. There is a recurring set of costs for discovery, problem determination, problem solution, re-integration, re-test, re-baselining, etc. Whereas in Inspections all identified defects are noted in the same proximate time period and the recurring costs are consequently reduced. Furthermore, when a defect is found during Inspections, we know where the defect is and often know how to fix it. Inspections find the defect and often the fix is apparent, but in test and with users often only the symptom is found.
- The increased labor hours required for fixing defects after the product is shipped is often due to loss of project team knowledge. Often another team will maintain the product and may need more time to discover what the defect is and what may have caused it before fixing it. Even when it is the same team, time has often reduced their ability to quickly determine the cause of a defect.
- When fewer defects enter test, the productivity of test improves; i.e., the costs of test are lower and the time to complete test is reduced.

Several studies have confirmed the reduction in project costs when defects are removed earlier. The relationship of costs to remove defects in production, test, and post-test was first shown in 1976 in Mike Fagan's article based on the study we performed in IBM during the first release of the telecommunications subsystem VTAM. [FAG76] Subsequent studies have continued to reconfirm these results with the same nominal cost relationship. [BOE81] [KIT86] [ACK89] [REV91] [DOO92] Figure 3 shows this relationship as increasing costs over time. Thus, if it costs on average \$100 to remove a defect in Inspections, it will cost \$1000 in test, and over \$10,000 when a user finds a defect.

The relationship is not always so orderly as 1:10:100, but the pattern is reasonably consistent as shown in Figure 3. The relationship has remained consistent since the first Inspections were performed.

Besides the costs to the project, we should also note that there is a cost to the customer for downtime, lost opportunity, etc. These costs while often transparent to the project can have a negative effect on future sales or contracts with the customer.

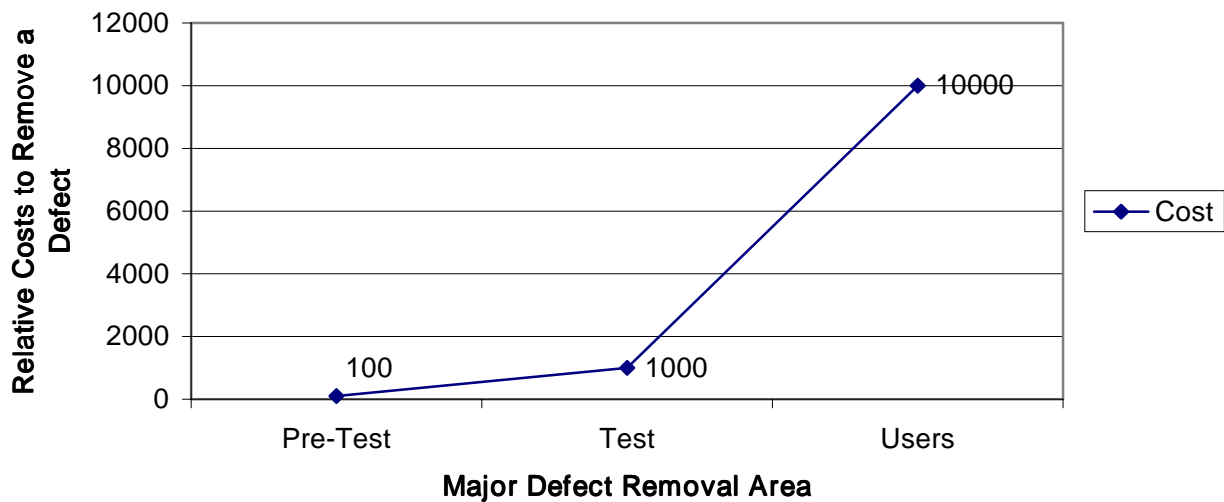


Figure 3 . Defect Cost Relationship

In ISO 9000, Inspections or *in-process inspections* are defined as one of the major elements required to be addressed to achieve certification, and Inspections can and have been used successfully in organizations not certified and perhaps not even certifiable to the standard. One does not need a formal quality system or SW-CMM [CMM93] maturity level to perform successful Inspections.



**Inspections have clear value
independent of any model or standard
for software development.**

When I am asked which Key Process Area in the SEI's Capability maturity Model (SW-CMM I think gives the most value or which should be the highest priority, I always respond with Inspections (Peer Reviews in the SW-CMM). I believed this to be true in the 1980's and I still believe it today. With an honest effort given to Inspections, we can open the eyes of management for more rapid process improvement. Inspections never fail to deliver as promised, or prove to be too difficult, when the preconditions are satisfied. We know how to make Inspections successful.

So Why isn't Everyone Using Inspections

If Inspections provide significant benefit, why has adoption been so slow? Why is not all of the software community using the method? The publicly published data twenty-four years ago clearly demonstrated their effectiveness, and recent reports continue to demonstrate their effectiveness. Why have Inspections not taken a firmer root and achieved wider use?

The reasons for this vary. I believe this is partially due to views that Inspections can only be done one way. This book will eliminate at least some of this misunderstanding. Another reason for the resistance to Inspections is the view that they are not easy to do well. There are factors, including social and psychological, that we must consider. Management often views Inspections as an added cost, when in fact Inspections will reduce costs during a project. We will see throughout this book how Inspections save both cost and time.

Since the inception of Inspections, with the advent of each new development tool or language, the providers and users seem to believe that Inspections do not add value or are less necessary. This is clearly not true, since programmers still make defects with these tools and languages. However there often is little documentation as to how well Inspections work in these new environments, so a false belief that Inspections do not apply may be propagated by myth. We will see later in the book that Inspections do apply in these development environments also.

Because of these and other factors, shortcuts in the Inspection process have been taken without any proof that the change is an improvement. Some of these homegrown changes cause the Inspection process to be less effective. In turn the value of what was defined as the Inspection process is diminished. Thus failure becomes a self-proving "fact."

Inspections may not be the most enjoyable engineering task compared to designing and coding. Also, programmers are very possessive about artifacts they create. Inspections are labor intensive and low tech, but they do work.

Do Good Programmers Make Defects?

How many times have you been convinced, absolutely convinced, that you had either written a defect-free program or had "tested out" all of the defects only to sadly discover that there was yet another "bug"? One has to wonder how we as programmers have been able to repeatedly survive this cruel onslaught to our egos. I am not talking about the slovenly programmer who in some instances seems to develop as many defects as lines of code (LOC) in a program or paragraphs in a document, or about the programmers who have found themselves in a schedule crunch and don't have enough time to develop their best work. Rather, I am talking about that qualified programmer (you and I are probably good examples; well, at least you), who does have sufficient time to produce a defect-free piece of work. Yet, after the program is delivered to test or to the user, defect fixes have to be applied to the work product. I am not going to try to come to terms here with why this may happen, for there can be many systemic reasons. Let it be known for now that good programmers do indeed make defects, and if this is true for good programmers then it must follow for all others in the population of programmers. All programmers can and do want to learn from their injected defects. We will see later in the book how both Inspections and Defect Prevention are natural combined processes for programmers to use, when they are adapted for effective success.

Every organization or company hires the best people. At least that is what we hear. One has to wonder where the worst people go. Some of them exist in every organization. Most cultures will not easily accept the reality of a natural distribution of capability and talent. It seems to be almost a taboo subject. Nonetheless it exists in every organization. What we should want to do, and what the SW-CMM enables, is to achieve a higher maturity and improved capability in software processes for all people in an organization.

People and processes become more capable as an organization moves up the SW-CMM levels. An example is shown in Figure 4 for the Inspection process from organizations I have come across over the past twenty-eight years. This figure represents by example that as an organization moves up the SW-CMM ladder, the effectiveness; i.e., the percentage of defects removed with Inspections increases with each level attained. Improved effectiveness should also be seen in other processes as an organization's maturity grows, but for Inspections it is easier and faster to see.

SW-CMM LEVEL	EFFECTIVENESS
1	<50%
2	50-65%
3	65-75%
4	75-90%
5	90-100%

Figure 4. Inspection Effectiveness and Maturity Levels

Effectiveness and Efficiency

Let us now take a look at the distinction between effectiveness and efficiency. They are different and both have business; i.e., economic, value to organizations practicing Inspections.

Effectiveness

Simply stated, effectiveness of Inspections is the percentage of defects removed by Inspections compared to the total sum of defects eventually found by Inspections, test, and in use by the customers. Data on the customer-found defects may require a longer period of time to acquire and may never be complete. In some cases the best we may be able to do is to extrapolate from a shorter period of customer use and assume a ratio for the entire set of defects that remain latent in the product.

For example, taking the first 3-6 months of customer use and resultant defect data and applying a model of distribution based on history in the company or organization for similar projects, then extrapolations of the total shipped defects can be made. It can never be precise, but it can be a reasonable calculation. The goodness of the extrapolations is dependent on the history available from comparable products and use of the products by customers.

Regarding the true volume of latent defects shipped with a product to users, in most cases this can never really be determined. We do not yet have the ability to decisively determine the real number of defects shipped with a product. We can project total defects based on other characteristics in the process or we can assume zero defects based on other process data and patterns, but we cannot prove it. For example, we can analyze error depletion curves prior to delivery to make a prediction of latent defects after delivery. I discuss more on effectiveness in Chapter 9.

Efficiency

Efficiency of Inspections is represented by various cost relationships; e.g., :

- \$ spent / defect found in Inspections
- \$ for Inspections / total project costs; this is a subset of Cost of Quality (COQ) measures
- \$ ratio of cost of finding defects in Inspection to cost of defects found in test and by the customers
- hours spent / defect found in Inspections

The use of the terms *effectiveness* and *efficiency* of Inspections including the relationship between them is not consistently applied in the software community. Some of this misinterpretation derives from Fagan's article where he refers to effectiveness as *efficiency*. This was because a focus at the time was on costs.

If we go back to first principles; i.e., use Inspections to find defects earlier because the costs for defect removal are reduced, then this is a measure of the efficiency. Defects are a cost driver to a project; i.e., the more defects the higher the cost to remove them. Therefore, the lower the cost to remove defects the more efficient we are.

Effectiveness has a relationship to efficiency in the project; i.e., if Inspections find more defects as a percentage of the total found and the cost to remove defects with Inspections is lower, then a higher Inspection effectiveness should be desirable since it drives the costs lower for defect removal on a project and hence also improves efficiency of the project.

While Inspections have historically proven to be more efficient, the technology of producing software is changing, so too may the understanding of where and how to best remove defects. Some defects may not only be cheaper to find in a test environment, but some may not even be able to be reasonably found with Inspections.

For example, the use of visual modeling tools and code generators; e.g., Visual Basic, PowerBuilder, JBuilder causes us to re-think what is the best way to remove defects. We need to revisit whether we should still read the code as in traditional Inspections or to map the code while reading the screen while executing the code. Mapping is a technique of keeping two documents in synch while reading the primary document during the Inspection.

1:1 Inspections

1:1 Inspections occur when there are only two participants in an Inspection, the Producer and the Inspector/Moderator. If you can perform a 1:1 Inspection you can typically save about 50% compared to the traditional Inspection. I discuss more about 1:1 Inspections in Chapter 14. Table 1 shows how the responsibilities for process activity change when a 1:1 Inspection is used and who performs the Moderator's tasks. In this table PL is Project Lead, P is Producer, and I is the inspector.

Moderator Task	Required?	Who Performs
1. Inspection Scheduling	Yes	PL
1.1 Determine need for Overview	Yes	PL & I
1.2 Determine Inspection team	Yes	PL
1.3 Ensuring availability of materials	Yes	P
1.4 Assigning roles	NA	NA
1.5 Chunking materials	Yes	PL & P
1.6 Defining activities schedule	Yes	PL & P & I
1.6.1 Overview	Optional	PL & I
1.6.2 Preparation effort	Yes	PL & P & I
1.6.3 Inspection Meeting duration	Yes	PL & P & I
1.6.4 Analysis Meeting	Optional	PL & P & I
1.6.5 Logistics	Yes	PL & I
2. Overview	Optional	P & I
3. Preparation	Yes	P & I
4. Inspection Meeting	Yes	P & I
5. Data Recording	Yes	I
6. Analysis Meeting	Optional	P & I
7. Rework	Yes	P
8. Follow-up	Yes	P & I

Table 1. Tasks Assignments in 1:1 Inspections

We see from this table that, while there is only one inspector, all Inspection activities and tasks are still performed.

Can Inspections Replace Test?

As noted earlier, Inspections are low tech, labor intensive, and rarely fun. These among other factors cause people to question the value of Inspections. A frequent challenge is put forward that some form of testing; i.e., unit test, will be just as effective and efficient or that Inspection cannot catch all defects.

I previously discussed that the solution for a defect is usually evident when it is found during an Inspection, thus the costs for repair are minimized. Tests after unit test do not make readily identifiable the area requiring repair and thus the costs increase. The countless attempts to prove that unit test is more efficient have failed and I know of no study that has been repeated where unit test has been demonstrated to be as effective as Inspections in removing defects.

If it is accepted that Inspections have value, the next challenge voiced about Inspections is that unit test in combination with code Inspections will lead to better results. Again, every trial I know of has failed in this regard. Russell [RUS91] gives two other reasons for not testing before Inspections:

1. As Inspections require a motivated team, testing first may lead to a view that the code is reasonably stable and the team will be less motivated to perform the best Inspection.
2. With the investment of test the Producer may be less receptive to major rework on an “already-stable program image” which will also require retesting.

Ackerman found that the savings from defect detection costs in Inspections was 2.2 hours compared to 4.5 hours in unit test. A two to one savings is a good place to bank. In another organization he states a 1.4 to 8.5 staff hour relationship in finding defects with Inspections versus testing. [ACK89]

Weller [WEL93] states that there are disadvantages of inspecting after unit test:

- Unit test leads programmers to have false confidence that the product works, so why inspect
- It is a hard decision to inspect a large batch that has been unit tested and there may be the view that there is no longer time to inspect

He also gives reasons to perform Inspections first:

- You may actually be able to bypass unit test if the Inspection results are good
- You can recover earlier with lower cost to serious design defects found in Inspections versus unit test

Despite this (and other) clear evidence, to this day many will suggest that unit test should be done before Inspections. Always this has proven to be less efficient. When I run into people who are hard to convince, I suggest that they try a simple experiment to see what the data tells them. I remain fully objective and state, “If the data proves that unit test before a code inspection is cost effective, then by all means proceed.” Twenty-eight years of requests and I have not been given proof that unit test is less costly. This does not mean the experiments should never be tried again. I discuss more on this in Chapter 14. For now I’ll leave the door open suggesting that as technology evolves, so too might the best way to remove defects.

Will Inspections Ever Become Obsolete?

Sometimes I hope so. I believe a clear possibility exists for Low-level design and code, but not for all work products. The highest cost for Inspections is in Low-level design, and code, so a reduction in these areas is a good contribution to reducing COQ. I suspect that for requirements and key design documents we will need Inspections for quite some time. For other artifacts such as plans, test cases, documents, I think we will see a reduction in a need for Inspections as we improve the disciplines of the activities for creating these work products through training, processes, and tools.

One focus we should take is that when an Inspection finds too many defects, this is a signal to management to do a better job of managing the software processes and people resources.



**There is no reason why programmers
should be put into positions of generating
100’s of defects to be found by Inspections.**

If we continue to inject high volumes of defects that are found in Inspections, then all we’ve done is shift the defect removal cost from test to Inspections. This will reduce costs; this is important, but it is not enough. We can do much better, by developing products with few defects that go into Inspections. We really can do it right the first time, if we try.

Deming in “*Out of the Crisis*” states “Cease dependence on inspection to achieve quality. Eliminate the need for inspection on a mass basis by building quality into the product in the first place.” [DEM82]

Can we meet this objective in software or will we always be plagued as Deming goes on to say about exceptions where “mistakes and duds are inevitable but intolerable.” I am not sure I want to claim that the software engineering process is an exception and it certainly shouldn’t lead to duds. We can be better than that. We can have excellence in programming on a broader scale than exists today.



**Inspections should not be a requirement to achieve quality
and they are not required for all work products.**

Solo:Inspections

Since project cost is related to the number of inspectors, we should want to minimize the number involved in any one Inspection. In the original method 4 inspectors were the norm and the IEEE standard suggests 3-6. If we can reduce to 3 from 4 we save 25% on traditional Inspections; reduce to 2 as in 1:1 Inspections and we save 50%. If it is possible to perform an Inspection with only one inspector, we could save 75% of costs of Inspections.

How you might ask can we reduce to one inspector? Consider an Inspection where the Producer is only involved for an Overview, where warranted, delivery of the material, and resolution of defects. Let’s explore how and whether this can be successfully applied.

We saw in Table 1 that 1:1 Inspections follow the same activities in the traditional Inspections with some tailoring to ensure effective process performance with fewer participants. Needless to say when effectiveness is achieved, then efficiency will also be in evidence. Is the same effectiveness possible with Solo:Inspections as we are seeing with traditional Inspections? The answer as I have observed is Yes.

In the Solo:Inspections, the inspector logs on to his workstation and brings up the work product to be inspected. He has all required documents available through the organization’s intranet for ancillary and reference documents he may need. He has access to the appropriate checklist, standards, and forms to record any defects found. He can open as many concurrent windows or views with required documents as he needs to perform the Inspection. His time is recorded based on log-on time. When he is done, he submits his Inspection documents to the Producer, Inspection Coordinator, and Project Lead. He is then available to the Producer if there should be any questions.

Another benefit is that the Inspection can be for longer than two hours, as the inspector can start and stop, chunking as he feels necessary. In Solo:Inspections the inspector makes a commitment to the Project Lead for a completion time or date and this is what he then can concurrently manage along with other work in his queue.

Organizations that are using Solo:Inspections move to them when their traditional Inspections have already proven high effectiveness. Since there is feedback and coaching to the inspectors about their effectiveness during this transition, the Solo:Inspection effectiveness remains high.

I do not recommend that organizations move to Solo:Inspections in one quick jump. But I do recommend that all organizations give it serious consideration. Recall that the first focus will always be on effectiveness. Solo:Inspections are an improvement in efficiency. After the effectiveness is proven, then the organization should look at efficiency improvements such as Solo:Inspections.

Table 2 shows how the assignment of responsibilities changes in Solo:Inspections. But observe that all the traditional Inspection activities are still performed. This set of activities is fully aligned with practice as defined by IEEE but a key difference is that the team is just one inspector.

Note that the Producer may still be involved, but only sometimes for the Overview, when an Overview is warranted. The Producer may also be involved in the Analysis Meeting and the Rework, when there are defects to analyze and discuss. It is also possible that the inspector may have questions for the Producer during the Solo:Inspection, but these and even the Analysis Meeting do not need to be face-to-face. It is also entirely possible that the Producer will not be involved in the Preparation or the Inspection Meeting, when the inspector has no questions. The Producer and inspector should agree on their protocol for asking and responding to questions during the Solo:Inspection.

Another advantage is that the Producer and inspector can be in two different locations. The inspectors I have observed seemed fully comfortable with this remote approach. One might wonder that given the starts and stops whether the inspector lost continuity of thinking and if this affected effectiveness. All I can say is apparently not, since effectiveness was consistently high. The programmers in these organizations were fully enabled to perform the best of Inspections, not to “just do an Inspection”. It was expected and tracked to learn that they would be within acceptable limits for time spent and for defects found.

Moderator Task	Required?	Who Performs
1. Inspection Scheduling	Yes	PL
1.1 Determine need for Overview	Yes	PL & I
1.2 Determine Inspection team	Yes	PL
1.3 Ensuring availability of materials	Yes	P
1.4 Assigning roles	NA	NA
1.5 Chunking materials	Yes	I
1.6 Defining activities schedule	Yes	PL & I
1.6.1 Overview	Yes	PL & I
1.6.2 Preparation effort	Yes	PL & I
1.6.3 Inspection Meeting duration	Yes	PL & I
1.6.4 Analysis Meeting	Optional	PL & P & I
1.6.5 Logistics	Yes	PL & I
2. Overview	Optional	P & I
3 Preparation	Yes	I
4. Inspection Meeting	Yes	I
5. Data Recording	Yes	I
6. Analysis Meeting	Optional	P & I
7. Rework	Yes	P
8. Follow-up	Yes	P & I

Table 2. Solo:Inspection Activity Assignments

Solo:Inspections, to date, have been best applied for code and Low-level design work products, but the concept could work for other work product types. Some work products, such as requirements specifications, may never be able to use Solo: Inspections, but most effort on Inspections is not in these work products anyway.

Lastly, we should recognize that Solo:Inspections would permit that we could use experts who are not part of our project, organization, or even company to perform Inspections for us on some work products.

A project could even subcontract Solo:Inspections to qualified domain experts.

In Conclusion

I've discussed some evolutions in Software Inspections; e.g., 1:1 Inspections and Solo:Inspections. I've addressed why Inspections can reduce the Cost of Quality (COQ), which today has a benchmark of about 22% in software organizations. That's a far cry from the 65% still seen in many organizations. Much of this COQ reduction can be attributed to the use of Inspections, so why would any organization not exploit Software Inspections?

Bibliography

[ACK89] Ackerman, A. Frank, Lynne S. Buchwald, and Frank H. Lewski, *Software Inspections: an Effective Verification Process*, IEEE Software, Vol. 6, No. 3, May 1989.

[BOE81] Boehm, Barry W., *Software Engineering Economics*, Prentice-Hall, 1981.

[DEM82] Deming, W. Edwards, *Out of the Crisis*, Massachusetts Institute of Technology Center for Advanced Engineering Study, Cambridge, 1982.

[CMM93] Paulk, M.C., B. Curtis, M.B.Chrissis, and C.V.Weber, *Capability Maturity Model for Software, Version 1.1 (CMU/SEI-93-TR-024)*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, February 1993. Also see ww.sei.cmu.edu.

[DOO92] Doolan, E. P., *Experiences with Fagan's Inspection Method*, Software – Practice and Experience, Vol. 22, No. 2, Feb. 1992.

[FAG76] Fagan, Michael E., *Design and Code Inspections to Reduce Errors in Program Development*, IBM Systems Journal, Vol. 15, No. 3, 1976.

[KIT86] Kitchenham, B.A., A.P. Kitchenham, and J.P. Fellows, *The Effects of Inspections on Software Quality and Productivity*, ICL Tech. J., Vol. 5, No. 1, May 1986.

[RAD02] Radice, Ronald A., *High Quality Low Cost Software Inspections*, Paradoxicon Publishing, 2002, ISBN 0-9645913-1-3.

[REV91] Reeve, J. T., *Applying the Fagan Inspection Technique*, Quality Forum, Vol. 17, No. 1, March 1991.

[RUS91] Russell, Glen W., *Experience with Inspection in Ultralarge-Scale Developments*, IEEE Software, January 1991.

[STA02] Stephanie Stahl, editor, *Editor's Note Software Quality*, Informationweek, May 27, 2002.

[WEL93] Weller, E. F., *Lessons from Three Years of Inspection Data*, IEEE Software, September 1993.

© Copyright Paradoxicon Publishing, reprinted with permission

Classified Advertisement

The Advanced .NET Testing System (ANTS) is a load testing tool for use early in the project lifecycle. By simulating up to 100 concurrent users accessing .NET Web services or other .NET applications it is possible to find out whether they scale before it is too late. Find out how the application scales as the number of users varies.

<http://www.red-gate.com/ants.htm> or sales@red-gate.com

Advertising for a new Web development tool? Looking to recruit software developers? Promoting a conference or a book? Organising software development related training? This space is waiting for you at the price of US \$ 20 each line. Reach more than 23'000 web-savvy software developers and project managers worldwide with a classified advertisement in Methods & Tools. Without counting those that download the issue without being registered! To advertise in this section or to place a page ad simply send an e-mail to franco@martinig.ch

METHODS & TOOLS is published by **Martinig & Associates**, Rue des Marronniers 25,
CH-1800 Vevey, Switzerland Tel. +41 21 922 13 00 Fax +41 21 921 23 53 www.martinig.ch
Editor: Franco Martinig; e-mail franco@martinig.ch ISSN 1023-4918
Free subscription: www.methodsandtools.com
The content of this publication cannot be reproduced without prior written consent of the publisher
Copyright © 2002, Martinig & Associates