
Keane White Paper

Meet the New Tester

Laying the Foundation for
Preventative Testing Practices

Keane White Paper:
Meet the New Tester: Laying the Foundation for
Preventative Testing Practices



EXECUTIVE SUMMARY



About Keane

Keane is an IT services firm headquartered in the US with more than 12,000 professionals worldwide. For 45 years, Keane has been an Application Services specialist with distinguished project management credentials. Today, we offer a flagship suite of Application Services, as well as Infrastructure and Business Process Outsourcing solutions delivered through onsite, nearshore, and offshore resources.

Visit www.keane.com to learn how our projects, managed services, and outsourcing engagements deliver value for a range of businesses and government agencies.

Confidential

Copyright ©2010 Keane, Inc.

The concepts and methodologies contained herein are proprietary to Keane, Inc. Duplication, reproduction or disclosure of information in this document without the express written permission of Keane is prohibited.

TABLE OF CONTENTS

Abstract.....	4
System Testing: Necessary, but Already Costly	4
Testing in Development	6
Enter the New Tester.....	7
Unit Testing.....	7
Exploratory Testing	9
Automation Services	9
Making It Work	10
Skills and Qualities	10
Management Structure	11
Peripheral Advantages.....	12
Selecting a Partner	13
About the Authors.....	14

TABLE OF FIGURES

Figure 1 Managing Structure	10
------------------------------------------	----

Abstract

In 2002, The National Institute of Standards & Technology (NIST) estimated that the cost to fix software errors post-release was 30 times higher than if the errors were fixed during the requirements and analysis phases of development. Today, new data shows that it is 100 times costlier to correct errors after software has been released than it is during the preliminary requirements-definition phase. We can expect this cost to increase as software continues to be deployed to the front office, where customers interact with it (and experience its bugs) first hand, and as we move more into the world of shared services and service-oriented architectures, where business processes and systems are increasingly integrated and where reusable software modules are increasingly employed.

Clearly, the answer to reducing the high cost of defect management is to exercise more preventative approaches to testing rather than traditional detective methods. In other words, in order to reduce the number of defects migrating into the system testing phase, it is essential to test as early as possible during the development cycle.

This paper discusses the merit of conducting structured testing during the development phase and provides a plan for achieving this goal with the help of a new type of tester.

System Testing: Necessary, but Already Costly

It is often cited that the cost of correcting errors or defects increases tenfold for each phase in the software development cycle. Defects may be introduced to a system at various stages of its development lifecycle, but in a testing regime that is heavily dependent upon a postdevelopment testing phase, defects that are introduced at the earliest stages of the project are allowed to migrate into later phases until they are finally exposed during testing, or worse, in the live system. This syndrome is known as defect migration (see the “Defect Migration” sidebar on page 2).

Defect migration is costly because it can cost 10 times more to isolate and remove defects for each phase that they remain in the system. Placing the onus of defect detection entirely on late-stage testing leads to greater total development cost and significantly increases the risk of project overruns.

Whether a traditional waterfall or an iterative development cycle is used, typical software projects involve a system testing phase that follows the completion of coding. The principal purpose of this phase is to ensure that the whole system behaves and performs correctly in an environment that anticipates (or at least closely represents) production usage.

Even if system test coverage is comprehensive and a high proportion of latent defects are found, correcting them will consume far more resources than if the same defects were trapped during development. This is due to the additional cost of the defect management and triage process, which involves investigating test failures, isolating the defect, raising the defect report,

Defect Migration: The Snowball Effect

Defects are introduced into a system because a mistake is made at a certain phase of the system's design or development. There are various types of mistakes and phases of development in which they are made. Here are a few examples of mistakes that introduce defects:

- » A requirement was ambiguously specified
- » A technical design decision was flawed
- » A coding algorithm was incorrect
- » A fix for one defect found in testing caused another elsewhere in the system Customer Retention

When a defect enters the system in one phase but is not removed until a later phase (or not removed at all), it is known as "phase escapement."

While it is better to find and remove defects in the testing phase (rather than letting users discover them in production), it is best to remove defects in the same phase in which they are introduced, and therefore it is best to apply quality control to the products of all phases, not just developed code. The reason for this is that the cost of defect removal increases dramatically for each phase that the defect remains.

The quality of the products from one phase directly affects the quality of those in the downstream phases. Furthermore, the impact increases as a defect escapes into later phases, because the size of the products of each phase increases dramatically. For example, a system may have hundreds of documented requirements, which leads to thousands of items in a technical specification, which leads to hundreds of thousands of lines of developed code, which leads to millions of lines of installed code.

If a mistake is made that leads to a defect in a single requirement, this can typically have a 10-fold impact on the design and a 100-fold impact on the code. If such a defect is not discovered until the test phase, it could therefore require changes to hundreds of lines of code. If, however, the same defect were found and corrected during the requirements specification phase (preventing the phase escapement), then the cost of correcting the requirement would be negligible, since design and coding would not have been affected.

prioritizing corrections, assigning the correction to a developer, reproducing the problem, fixing the code, notifying testers, retesting the fix, and closing the report (see the "Defect Management and Triage" sidebar on page 5.)

System testing is a valuable activity that is meant to find the sort of defects that are not easily detectable in a development environment, such as supported hardware or platform compatibility, integration with

other systems, end-to-end usage scenarios, behavior with real data, performance under load, and so on.

So, system testing is necessary, but it is also costly. Test rigs and environments are complex to set up, external system interfaces must be connected or simulated, tests are generally longer to run, and in many cases expert business users are required to assess results or investigate anomalies.

Ironically, the more effective the system testing, the more difficult it is to keep to the testing schedule. The greater the number of defects encountered during system testing, the greater the triage burden and subsequent pressure to run extra test cycles to allow for retesting and regression checks.

Too often, valuable system testing time is eaten up with finding and resolving basic functional defects that could (and should) have been detected by unit testing in the development phase. Furthermore, many such functional defects effectively block access to certain functions or areas of the system under test. These blocking defects will further impact the testing schedule as they may render other valid and important tests unable to be executed until the blocking defect is fixed.

Clearly, in order to avoid this double-whammy of high costs and unpredictable time-frames in the system test phase, it is necessary to improve the quality of the system under test before it reaches this phase. Defects are an unfortunate fact of life on a development project, but their impact on the project schedule (and stakeholder confidence) can be vastly reduced if some effective “border control” is applied to prevent them migrating.

Some of the most cost-effective methods to minimize defect migration from the development phase are:

- » Introduce test-driven development (automated, programmatic unit testing by developers which applies to all new code).
- » Provide automated functional regression tests to be applied to daily software builds.
- » Perform directed, exploratory manual testing by an experienced tester on the software under development.

All of these methods require an awareness and respect for testing within the development phase, which unfortunately most teams are missing. This can be resolved by adding a new member to the development team responsible for ensuring the quality of testing activities in the development phase. We will discuss the role of this new member later on.

Testing in Development

There are two main types of testing that can be carried out during development:

- » **White box unit testing:** intended to confirm the programmers’ expectations about the code they write
- » **Black-box functional testing:** intended to confirm the user’s expectations about the system’s functionality

Until the relatively recent advent of test-driven development, unit testing by developers has been something of a forgotten art. It is often regarded as no more than an unpleasant chore that is demeaning to the professional developer and a waste of time. As a result, unit testing is typically unstructured, ad-hoc, undocumented, and therefore provides low information value about quality and risk.

Unit testing and other quality control measures in the development phase are a great opportunity to restore quality, stability, and value in later testing phases. This opportunity stems from the difference in time and cost to fix defects in different phases. A defect discovered in development can be nipped in the bud and resolved without the need for the high visibility

and cost overhead of the whole defect management process.

If developers are effectively testing their work as it is developed, any “additional” time spent on defect correction is reflected in the development time for that task, which is precisely as it should be; the feature isn’t done until it’s done correctly, after all. The development manager can monitor any schedule slips at the task allocation level, and remove low-priority tasks and features if necessary. This is far preferable to proudly announcing that all features are complete, only to discover during testing that a whole lot of remedial work is required.

Enter the New Tester

“OK guys, meet the newest member of the team. She will sit right here with us, will attend all the daily stand-up meetings, and will take part in the coffee-round. Oh, and she’s a tester.”

The hardest task that our new member on the development team will face will be to gain the trust and acceptance of a team of people that might initially consider testers to be a completely different species. That’s why she needs to be no ordinary tester, but a new breed of individual who is capable of proudly carrying the banner of quality while understanding the realities of pressures on a development team. The tester should not view her role as going behind enemy lines; rather, she should see herself as an ambassador of the testing group, preparing the way for smooth negotiations later on.

A tester who is embedded on the development team has a number of activities to carry out, but the role is

essentially this: to assure the quality and functionality of the code produced, so that subsequent testing can reliably focus on proving business processes rather than application functionality.

The primary tasks for the new tester are:

- » Assisting developers with unit test design
- » Performing exploratory testing of new features
- » Providing test automation services and expertise to the development team

Unit Testing

The developers will primarily be responsible for their own unit testing, but the tester assists with this task to raise the quality of unit testing and bring the team up to a competent standard. To use Erich Gamma’s phrase, the idea is for the tester to get the team “test infected.” (Gamma is co-author of the landmark book, *Design Patterns*.) In other words, to instill a permanent change of mindset that ensures the programmers don’t just consider coding without good, effective unit tests to be bad practice, but to see it as plainly unsafe.

Unit tests are necessarily small, atomic tests that are closely associated with a particular method, object, or code module. Unit tests can be manual, but this is generally not a recommended strategy. Instead, effective unit tests should be programmatic, self-checking assertions of expected results given fixed inputs. The tests themselves should be authored in the same language as the application under development (so the developer does not have to

Defect Management & Triage: A Risky Business

Defects discovered by testers during system testing are subject to a defect management process. Because the responsibility for detecting and fixing defects are split between testers and developers respectively, it is important that a process is followed that ensures defects are carefully tracked and managed to closure. The stages of a defect's lifecycle usually include:

New: Initial state of a new defect report. A test manager usually reviews each new defect report and may reject it if it is not considered valid. Otherwise it is promoted to the "open" stage.

Open: The defect has been reviewed or approved but has not yet been corrected.

Assigned: The defect has been prioritized and assigned to a particular developer.

In Progress: The assigned developer is working on the problem.

Retest: The developer has provided a fix that is ready to be retested. The tester can reopen the report if the problem still occurs or close the report if the problem is fixed.

Closed: The tester has confirmed that the fix has resolved the defect.

The names of these stages may vary according to the defect management tool used, but the lifecycle generally follows this pattern.

A critical element of the process is defect triage. This typically requires a committee of representatives from testing and development (such as the respective managers or team leaders), the project manager, and possibly marketing or customer representatives. This committee must assess all unresolved defects, with the aim of defining the priority for fixing each defect and setting targets for correction. This decision process will take into account a number of factors, such as the possible impact on a customer, the current impact on testing, and the anticipated time and cost of correction. Depending on the deadlines and resources available, the decision options regarding some defects may include taking no action at all.

Triage is expensive. The more important the system under test, the more senior the stakeholders must be that take part. In the thick of testing, triage meetings should take place daily. The longer the list of open or new defects, the longer these meetings will take.

Triage is also risky because it forces people to make difficult decisions under pressure, with possibly far-reaching ramifications. There is a natural tendency as shipping deadlines loom during end-game testing to leave the code alone for the sake of avoiding further instability or regression; therefore, many known defects may be shipped with the product. This is damaging for the reputation of the company and negates some of the value of the time invested in testing.

Furthermore, what may appear to be a relatively innocuous defect or failure in the context of one test may in fact be a symptom of a more serious fault in the system that has not been properly investigated.

switch tools), and the test code must be subject to the same version controls as the main code. All that is needed to manage the test execution is a framework such as the freeware xUnit family.

Developers will actually feel at home writing programmatic xUnit tests, because it doesn't feel like testing, it just feels like programming using a very elegant tool. However, once the first flush of excitement has passed, developers will find that they

want to write good tests rather than just any old test, and they may not have the right grounding in the principles of testing to do so. The new tester is the agent to help. The tester can pair up with different developers in turn, jointly assessing the effectiveness of their tests and discussing how they can be improved. The tester can provide pointers on whether a test is sufficiently isolated, self-describing, atomic, and so on. The tester can suggest additional variations on tests for other classes of input data. Have all the error conditions and exceptions been checked? Are default values handled correctly? Do we have inputs for all boundaries?

Exploratory Testing

During development, the embedded tester will subject all new features to “black-box” or functional testing (though the tester’s familiarity with the development of the system will enable her to apply greater insights into testing, so the box is more like smoked glass than black).

When the tester performs functional testing in development, she is acting as a surrogate customer of the application and also a devil’s advocate, thinking the way a user would think in terms of usage scenarios, but exercising the application from the basis of healthy scepticism rather than acceptance.

Functional testing of software that is still under construction obviously calls for a somewhat different approach from the more formal testing usually applied at the system testing phase. The tester must be prepared to be dynamic with her planning and much more constructively engaged with the developers

about reporting and investigating unexpected behavior.

Exploratory testing is a manual testing technique that is particularly well suited to early functional testing during development. The intention is to cover new ground quickly, seeking out likely sources of errors in functionality that has not been heavily exercised before. It is not exhaustive, but is likely to uncover enough issues to keep developers busy until formal tests can be scripted (or if no issues are found, provide confidence that scripted testing is worthwhile). The subject of exploratory testing is well covered by programmer and SQA guru James Bach at www.satisfice.com.

Automation Services

The final role for the tester is to help herself by seeking to automate any activity that is worth repeating often. She should be able to identify the areas in which automation will reduce the effort of keeping quality standards high. There are a number of tools available that will bring value to different activities. The development team as a whole must be willing to work together to implement and use these tools, but they will look to the tester to provide the inspiration.

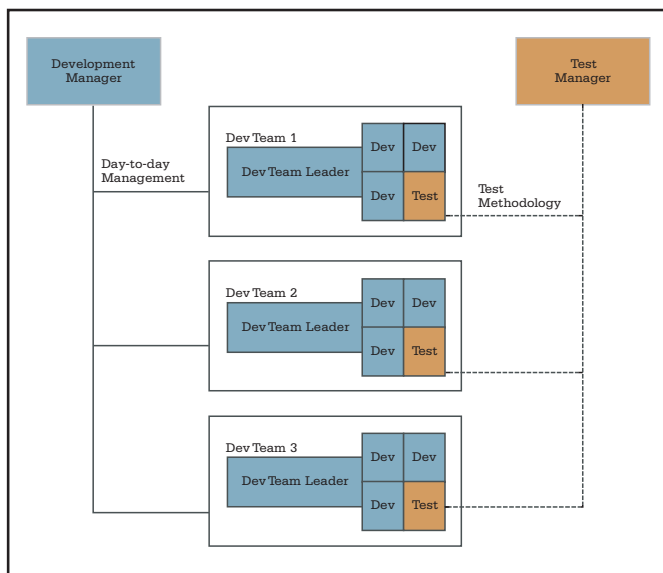
Code coverage analysis tools are an invaluable resource, providing a good indicator of the extent to which a test suite exercises the code base. The tester can be responsible for setting up a coverage analysis tool and provide measurement of the coverage levels of unit tests on daily builds of the software. This information can be fed back to the developers who can

increase the coverage of their automated unit tests accordingly.

The xUnit test framework mentioned earlier is a tool of the trade with which the tester must become familiar. Apart from being the natural execution tool for the growing repository of developer unit tests, the framework can also be used to invoke other automated tests that the tester may write for her purposes. It can be used to execute an automated regression test suite on a nightly build, ensuring that all checked-in code has left the system in a healthy state.

Functional tests that preserve the correct behavior of the system against regression errors can be automated. There are a number of commercial tools for this, as well as some open-source frameworks, such as Fit.

Figure 1: Management Structure
While the "new tester" sits among the development team and takes day-to-day direction from the development team leader, she is answerable to the test manager.



Any automated tests should become part of an automated testing process that can be executed frequently, such as after a nightly software build on an integration or smoke test machine.

Making It Work

We have described above the role that a tester can fill when she is embedded in the development team. But what skills and qualities must the tester possess to be successful? And what changes to the organizational structure are required to reap the full benefits of this new role?

Skills and Qualities

The role of the embedded tester is no small one. A test manager will do well to choose carefully the right candidate to be successful within a development team. The tester must be a personable, communicative person, able to state her case dispassionately but confidently to possibly defensive developers.

The tester must also be technically confident and ideally should be capable of reading and understanding program code, even if not required to write it. The tester must also have the confidence to hold her own in discussions about the readiness of the software and must not bow to pressure from the development manager to let through sub-standard code.

An active interest in the tools of the trade is a must. Developers will be willing to use tools to support quality initiatives if they can be convinced of their worth and effectiveness. The more competent the tester is with these tools, the more acceptance she is likely to win from the developers.

Automation skills are a definite bonus. The ability to code unit tests, write batch files, or debug Perl scripts will encourage creative solutions to embedding quality processes, as well as earning hefty credibility with the developers.

Management Structure

So where does our new tester sit within the management structure of the project? The key point to remember is that these people are dedicated testers. They may sit among the development team and they may be executing something between unit tests and system tests, but they must be answerable to the test manager. In this way the tester has a responsibility to uphold the testing ethos and to ensure that these aims are not subverted by conflicting demands of the development team (for example, meeting a delivery deadline).

As with all other phases of testing, development testing must adhere to a set of project standards. It is likely that these standards will differ from those for system or user acceptance testing, but the basic principles that apply to all testing are still relevant: the testing must be visible, accountable, and documented in sufficient detail to provide confidence in the testing carried out and, if necessary, to allow handover to another experienced tester.

But these testers are not some special presystem test team, sitting apart from the developers; these are embedded testers. The testers must be co-located with the rest of the development team and get involved in the day-to-day activities of the developers, attending the developers' planning, design, and progress meetings. Even more valuable, by sitting with the developers, the new testers will hear all the informal information that passes around the team, much of which will contain vital information about the system behavior and pointers about where to direct testing.

Although answerable to the test manager, the new tester will take direction on a day-to-day basis from the development team leader. Only the development team leader knows which packages are actively being worked on and which are ready to be tested, and can steer the work of the tester appropriately (see figure).

The delivery out of the development team is now dependent on the tester successfully passing her tests. But the tests to be executed are directed by the test strategy or methodology, which is the responsibility of the test manager. This means that the development team leader appears to be dependent on something that is outside his control (for example, he cannot affect the tests to ensure that they meet his delivery deadline). On occasions this can be frustrating for development team leaders and often they will exert undue influence to get tests changed or removed. For this reason the new tester needs to be a senior tester with a strong personality, who is comfortable defending her testing principles. And of course she needs to be backed up and supported by the test manager.

Of course, it is not really true to say that the development team leader is responsible for something that is outside his control and he shouldn't find having an embedded tester is at variance with his objectives.

His goal is to delivery timely, high quality software, and the new tester is helping him to do this by providing accurate and unbiased feedback. The development team leader should welcome the new tester as a valuable resource for the team.

Peripheral Advantages

We talked initially about the cost savings that can be made by identifying defects earlier in the development life cycle. Having embedded testers is a key means of realizing these cost savings; however there are a number of other benefits that will be enjoyed.

Bridging the Divide

In some organizations there is an apparent conflict of interests between developers and testers. A divide, both physically and emotionally, can spring up between the two parties and a degree of finger pointing on both sides can foster resentment between them.

By embedding testers in the development team, barriers are broken down between the two teams. The natural interactions that happen as a result of sitting among the developers will foster a spirit of understanding and create personal relationships between the two teams' members. This, in turn, promotes co-operation between the teams and can help the project work as a coherent unit.

Sharing Responsibility for Quality

The embedded testers serve also to raise the profile of testing within development. By making each

development team responsible for the quality of their software, where that quality is independently verified, means that each team has to ensure that the software it produces functions correctly. It prevents a "that'll do" mentality as developers cannot throw the code over the wall until it has correctly passed its tests. In essence, it keeps the developers honest.

Making Exit Criteria Mandatory

The new testers' tests are, by their very nature, the exit criteria from development into system testing. In many organizations, exit criteria for each phase are elaborately set out during the planning phase only for them to be ignored when the project gets underway. With the independent stamp of approval required from the embedded testers to exit development, the exit criteria are a visible and integral part of the process. The project management can choose to disregard them, but it cannot pay them lip service and just forget about them.

Streamlining the Feedback Loop

By embedding testers within the development team the develop-test-fix cycle becomes a much tighter loop. The testers are able to provide very swift feedback to the developers, and the developers can provide fixes very quickly, without the need for extensive change control procedures. Normally only unit tests enjoy this level of near instantaneous response, but unit tests are not produced by professional testers and are not designed with the same attention to detail and the business-oriented mindset that professional testers' tests are. The result is cleaner and more robust builds, which give the system test a much greater chance of success. Given that the system test phase is the one that is most commonly cut short by fixed delivery deadlines, any

improvement in quality going in to system test will reduce the total cost of testing and result in fewer production problems.

Selecting a Partner

Many firms choose to work with a testing partner capable of driving change in its testing practices. Consider the following when selecting a testing partner:

- » **Business orientation to testing:** The ideal partner must understand the strategic business reasons for testing and allocate resources accordingly. Risk-based testing approaches ensure that testing initiatives target areas of greatest business importance.
- » **Vision of improvement beyond:** finding defects: A high-value testing partner uses the knowledge gained during testing to improve development processes to prevent similar defects from being introduced in the future. An emphasis on requirements definition and validation of specifications ensures a linkage of requirements to desired business outcomes and results in completed systems that more closely meet business objectives.
- » **Production experience:** Real-world experience in supporting production applications is essential for understanding where and why to apply testing to prevent defects from impacting service and business performance.
- » **Willingness to commit to results:** A trusted partner is willing to be held accountable for its

performance through metrics and service level agreements. This commitment includes the quality of its results, the productivity of its teams, and its ability to leverage knowledge gained during testing to continually improve development processes.

- » **Project management:** A properly executed testing program involves considerable coordination and communication to keep business objectives in sight while ensuring the flawless execution of countless details. A prospective partner's project management expertise is as important as its technical testing knowledge for guaranteeing the success of its delivered services.
- » **Strong testing methodology:** Ensure that a prospective partner's methodology is available for examination, follows industry standards and best practices, and is actually put to use on its projects. Keane's lifecycle management approach ensures that testing begins at the start of the development process and becomes an integral activity as development progresses. As a result, we are able to focus on quality and defect prevention while reducing development time and cost. For more information on Keane's QA & Testing service, read "An Ounce of Prevention: The Importance of Testing": available at www.keane.com/whitepapers.
- » **Solid testing practice:** To effectively deliver its services, a testing partner must have the training experience, resources, and global delivery capabilities to apply its testing methodology to any tool or infrastructure environment. Keane's QA & Testing Service is built on 40 years of software development experience and adheres to rigorous CMMI standards. Our strong industry

partnerships enable us to deploy the best solution to the problem at hand.

- » **Broad range of delivery options** : Every testing effort is different. The ideal partner offers flexible delivery options that can be tailored to the management, resource, and execution needs of an individual engagement. Keane's Global Delivery Model provides clients with delivery alternatives integrated by a common management structure. Options include: on-site at the client location, off-site at Keane branches, near-shore at our Advanced Development Centers in Canada, and offshore at our Advanced Development Centers in India.

Whether you choose to work with a testing partner or go it alone, the key to reducing the number of late-phase defects — and the high costs associated with them — is to integrate testing best practices into the development phases. For many firms this represents a seismic cultural change. Success depends on bringing in a new type of tester with the technical skills and personal characteristics that can help lay a foundation of preventative testing practices.

About the Authors

This paper was authored by David Evans and Nick Pointon, with contributions from Samuel Waithe.

David Evans and Nick Pointon are directors at Cresta Group Limited, an independent software testing consultancy headquartered in London. Samuel Waithe is the director of global strategy for Keane's QA & Testing Service.

The body of this white paper was originally published in The Professional Tester Magazine in March and June of 2005 and has been republished by Keane with the authors' permission.

In June 2005, Keane acquired Cresta Testing Inc., the US affiliate of Cresta Group Limited.
