



# Overcoming The Developer Testing Paradox

## *The Key to Securing the Future of the Testing Profession*

BY BERT JAGERS

TESTING CONSULTANT  
CTG BELGIUM

### Abstract

*The directions the testing profession can evolve to, depends largely on what customers are willing to pay testing professionals for. Looking deeper into what the customer wants, we can see that like many products and services, it actually comes down to the need for good product quality, as fast and as cheap as possible. As it is faster and cheaper to fix defects earlier in the development lifecycle, then why is development testing not gaining popularity each day. Why is developer testing not a common practice?*

*Since a Developer Testing Paradox is currently keeping us from testing as early as possible, we must investigate what the possibilities are to compensate for this paradox. Is it to review requirements and design better? Can we do it by infiltrating the development team? Or is it by evolving our testing profession more towards specialization?*

A SPECIAL CLIENT REPORT

---

Copyright ©2008 Client Confidential: A special report for clients only. The material covered in this report is for guidance only. Application and implementation guidelines, advice, and consulting are available.

*Copyright ©2008 Client Confidential: A special report for clients only. The material covered in this report is for guidance only. Application and implementation guidelines, advice, and consulting are available.*

*Boehm's cost of change curve implies that it is less expensive to identify and fix a defect during component and component integration testing than during system (integration) testing and acceptance testing.*

*Therefore, testing is not something to be exclusively left to system or acceptance testers, or to be done only at the end of a project.*

## Introduction

When considering the future of software testing, the issue more or less comes down to one critical factor; what does the customer or employer want to pay for? After all, professional testers can only afford to continue being testing professionals if they get paid for their work. But what is exactly “what the customer wants”? And how can we deliver? Is it good product quality, as fast and as cheap as possible?

## The Developer Testing Paradox

### Boehm's Legacy

Since Boehm published his famous “cost of change” curve in 1981 (Boehm, 1981), the following economic fact has become more and more widespread: “it is much cheaper to fix a defect early in the development lifecycle” (figure 1).

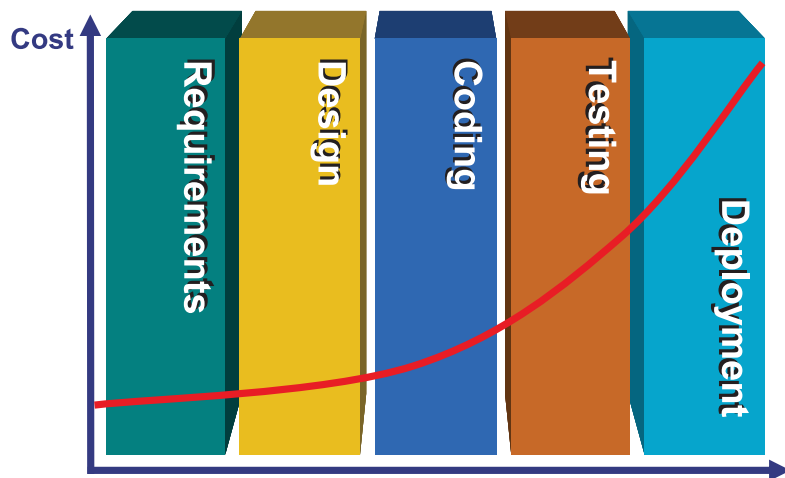


Figure 1: Boehm's cost of change curve

Within the V-Model, component testing is the test level where individual software components are tested, while during component integration testing, interfaces and interactions between these software components are tested (ISTQB, 2007; IEEE610, 1991). These two test levels are also generally known as developer testing or unit (integration) testing.

Boehm's cost of change curve implies that it is less expensive to identify and fix a defect during component and component integration testing than during system (integration) testing and acceptance testing.

Therefore, testing is not something to be exclusively left to system or acceptance testers, or to be done only at the end of a project.

By integrating testing directly into a development process, you can produce more robust and error-free code (Gunderloy, 2004).

*The root cause of most software project failures, and of the poor general health of most software, is the lack of early-stage unit testing (SAVOIA, 2005)*

## **Identifying the Paradox**

If the above is true, then why hasn't developer testing, become standard by now? The root cause of most software project failures, and of the poor general health of most software, is the lack of early-stage unit testing (Savoia, 2005).

So unit testing is considered to be the best option to improve software quality, and the benefits of it seem to be clear (Binstock, 2008b):

- Defects are identified nearly immediately, and they are also resolved more quickly. As a result, you can be more confident in the project schedule.
- Through unit testing, you can get better code-level documentation.
- Unit testing results in better coding and increased confidence in the code.
- The cost of solving defects found during unit testing is significantly lower.
- Because the primary tools for unit testing are free and open source, the initial costs for trying and adopting unit testing are low.
- Unit testing can be used with any form of development philosophy, so adoption is non-disruptive.

Yet, most companies are not eager to start using unit tests, condemning it to remain more of an obscure niche within the growing domain of software testing. This problem has already been identified as the Developer Testing Paradox (Savoia, 2005). The Developer Testing Paradox is recognized as one of the major paradoxes within the testing profession as we know it (Van Hese, 2007)

To keep your software healthy, you need an immune system that only thorough unit tests can provide. Many developer testing efforts start with great enthusiasm but end up with less than stellar results, partial adoption, or total abandonment. This is why regular, ongoing developer testing practices are the exception rather than the rule, and why there is a developer testing paradox in the first place (Savoia, 2005).

Several reasons for developer testing being an exception rather than a general practice have been identified. They can generally be grouped under the following problem areas:

- a lack of evangelization within the non-Agile communities (Binstock, 2008a):
  - Commercial Unit Test Tools companies see their market diminish (Koch, 2008)
  - There is a limited set of open source unit test tools available. Not every coding language is supported yet
  - Developers are insufficiently educated on the use of unit testing
  - Hardly any literature on unit testing is published (one book a year)
- an overabundance of evangelism within parts of the Agile community (Binstock, 2008b):
  - A segment of the Agile development community has adopted unit testing in a radical way, insisting that tests be written before the code. This technique is also known as Test Driven Development or TDD (Beck, 2002). For many developers that evangelism has become so overbearing that it makes unit testing as a whole unappealing.

*Testing as such is not always very popular within development, and the ranks of developers and teams that actually practice unit testing are modest indeed, if not increasingly small.*  
(KOCH, 2008)

- the added value of unit testing is not always recognized (Binstock, 2008b)
- the cost of unit testing (Binstock, 2008b)
  - It takes time and skill to write good unit tests.
  - All developers need to do it, for unit testing to really deliver.
  - When a software base changes in an important way, the unit tests must be discarded or substantially rewritten.

Most organizations that see past these concerns and actually try unit testing usually find the return is well worth the cost, and typically expand their unit testing efforts (Binstock, 2008b).

### **The Genetic Code of a Developer**

As described in the previous section, one of the costs of unit testing resides in the fact that all developers in the team need to do it. This in itself is a steep hill which unit testing needs to overcome in order to gain popularity.

Testing as such is not always very popular within development, and the ranks of developers and teams that actually practice unit testing are modest indeed, if not increasingly small (Koch, 2008).

Savoia divided developers into three groups based on the different genotype they possess, namely:

- **T1: Very susceptible to test infection.** Show them a single JUnit example and they get it immediately—and start using it regularly and with great fervor. When time pressure hits, they fight hard for enough time to test and would rather quit than produce code without tests.
- **T2: Somewhat susceptible to test infection.** Show them how to use JUnit/TDD, give them some encouragement and time to appreciate the benefits and rewards; eventually they will become regular users. When time pressure hits, however, they often revert to no-test mode without too much kicking and screaming.
- **T3: Immune to test infection.** There is no amount of showing and convincing that will get them to practice developer testing on their own with any regularity—if at all. Many of them would rather quit than have to write tests on a regular basis.

Savoia also stated that according to his experience, the T3 type was overabundant in the population (Savoia, 2007).

### **Developer Testing and Natural Selection**

The fact that the T3 type is so commonly predominant within the developer population might lie at the core of the Developer Testing Paradox. Savoia (2007) states that when developers become managers, they bring with them their attitudes toward developer testing. Therefore, a T3 manager is not going to provide an environment where the T2 developers (the ones that require some time and encouragement) can become test infected.

*As long as doing developer testing does not make a developer more “successful” than ones who don’t do it, developer testing will never become generally accepted.*

In evolutionary biology, natural selection is the process by which favorable genetic traits become more common in successive generations of a population of reproducing organisms, and unfavorable genetic traits become less common. This principle is not only valid within the biological universe but it is also applicable within lots of other different areas, like software testing. After all, the favorability of a trait depends highly on the environment an individual finds oneself in.

At this moment, the T1 genes are considered by many in the developer population as unfavorable traits. The current environment within most development teams or companies is so that doing developer testing will not make an individual (more) successful. Therefore, the T2 types will mimic the T3 behavior. The moment developers who do development testing are actually rewarded for doing development testing, and thus become successful, the T1 (and T2) genes will become a larger part of the overall population.

As long as doing developer testing does not make a developer more “successful” than ones who don’t do it, developer testing will never become generally accepted. The T3 individuals will remain the majority of the population as they can be as successful (or even more) as a T1 or T2 individual by doing less.

As long as this is the case, the first line of defense against defects will keep on failing or remain insufficient.

## **Coping with the Developer Testing Paradox**

### ***Reinforcing the Second Line of Defense***

Now, if a Developer Testing Paradox keeps project teams from finding defects as early and cheaply as possible, then what is the next best thing?

If the first line of defense is letting the team down, there are three possible options:

- Creating a new, even earlier, first line of defense, by conducting reviews
- (Black box) testing within the development team
- Reinforce and improve the second line, namely independent verification during system and system integration testing

### ***Reviewing***

Reviewing has been defined as an evaluation of a product or project status to ascertain discrepancies from planned results and to recommend improvements. Examples include management review, informal review, technical review, inspection, and walkthrough (ISTQB, 2007; IEEE1028, 1997). The objective is to find defects earlier in the life cycle and to remove the causes of the defects from the process.

Requirements of complex software applications are often negotiated through two concurrent dialogues, which continuously evolve throughout the project lifecycle. These dialogues are focused around two questions: “what do we need to build?” and “what can we build?” The quality of these dialogues often determines the ultimate quality of the constructed application (Aharonovitz, 2007).

*Reviewing of requirements and analysis documents (and even code) will certainly improve the identification of defects during a phase in which the cost for solving these defects is still rather low. Performing reviews will not completely compensate for a lack of developer testing though. Ideally, it should be considered as an extra testing activity alongside of developer testing, system testing, acceptance testing, etc.*

In 1984, James Martin stated that the origin of the largest proportion of defects could be traced back to the requirements and the design phase. Of all defects found during the requirements phase, 50% were the result of poorly written, ambiguous, unclear, and incorrect requirements. The other 50 percent of requirements defects could be attributed to incompleteness of specification. Only a small percentage of defects are actually introduced during coding.

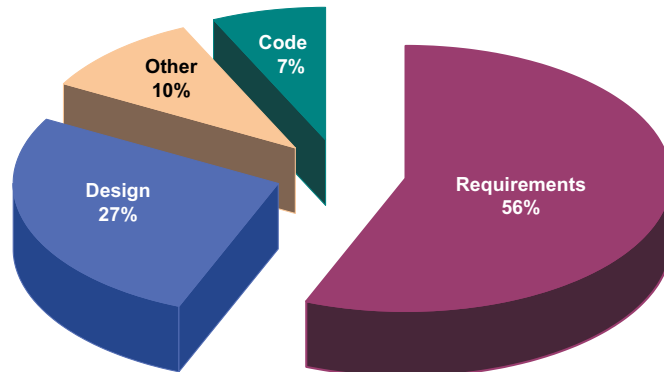


Figure 2: Distribution of Defects

When looking at Boehm's curve again, the benefits of reviewing are now very clear. Reviewing of requirements and analysis documents (and even code) will certainly improve the identification of defects during a phase in which the cost for solving these defects is still rather low. Performing reviews will not completely compensate for a lack of developer testing though. Ideally, it should be considered as an extra testing activity alongside of developer testing, system testing, acceptance testing, etc.

Conducting reviews provides several additional benefits:

- Defects can be identified and solved in a phase where it is still relatively inexpensive to do so.
- A large percentage of defects can be solved in the same phase they are created
- The test team gets involved earlier in the project
- Knowledge of the product can be gained in an early phase
- Several testing activities are removed from the critical path
- Stakeholders generally become more aware of the quality objectives

### ***Infiltrating the Development Team***

Too often, valuable system (and even acceptance) testing time is taken up with finding and resolving basic functional defects that could (and should) have been detected by unit testing in the development phase. This phenomenon has been identified as the process of Defect Migration (*Narayanan, 2002, Evans & Pointon, 2005*).

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.

*Testing small parts of the application allowed us to create very short cycles in which I identified defects, while developers were already working on solving some of the defects I previously encountered. Within the hour, a new release was created and solved defects could be retested.*

Evans & Pointon presented a new testing profile that functions within the development team. This approach looks to suppress defect migration as much as possible. *Evans & Pointon (2005)*

Within the development team, this tester should be treated as a full-fledged member. Yet the tester remains a representative of the system test team who reports to the test manager.

This 'embedded' tester has a number of activities to carry out, but the role is essentially this: to ensure 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
- Exploratory manual testing of new features
- Providing test automation services and expertise to the development team

### ***Infiltrating the Development Team: A Case Study***

During the year 2006, I took part in an interesting experiment at a customer site. The problem encountered at that site resided in a mutual feeling that the cycles of releasing for system testing, the actual testing and the correction of defects took too much time. The amount of developer testing that was done was rather low, resulting in the identification of defects during system (integration) testing that could have been removed by means of more thorough developer testing.

As part of the Test Competence Center, responsible for system (integration) testing, I was asked to temporarily join the development team in order to test small parts of the application in a manual black-box, exploratory way. Testing small parts of the application allowed us to create very short cycles in which I identified defects, while developers were already working on solving some of the defects I previously encountered. Within the hour, a new release was created and solved defects could be retested.

The key benefit was that defects were encountered during the development phase and solved quickly within that same phase. The quality of the system under test was much higher when it was released to the Test Competence Center for system (integration) testing, reducing the duration of the system test phase considerably.

## **The Evolution of a Tester**

### ***A Jack-Of-All-Trades is a Master of None***

A third possible and not unimportant solution on how to minimize the impact of the Developer Testing Paradox is the testing profession itself. How can system and system integration test levels be optimized to cope with the known lack of development testing?

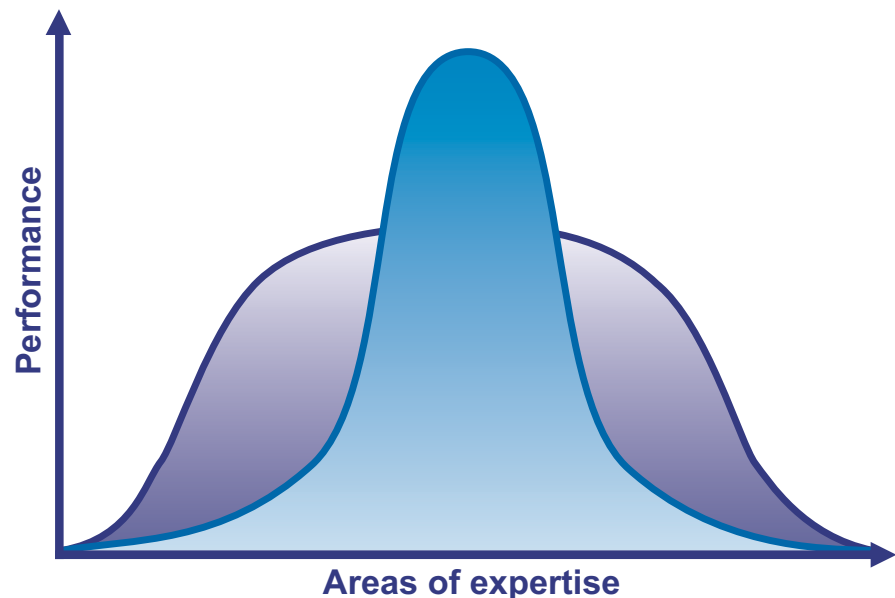
One of the questions one can ask is how do we as testing professionals get better or more efficient at finding defects.

*As testing becomes more and more mature as a profession, more and more niches are discovered and adopted by testing professionals. Over the years testing has become much more than just checking whether an application works.*

Does the profession benefit more from people who specialize in certain areas or from generalists who know some things about a lot of areas?

An interesting concept in this discussion is the “jack-of-all-trades is a master of none” principle (also known as the principle of allocation or the principle of trade-offs). This principle distinguishes between two types:

- **The Specialist:** Someone who is an extremely specialized in a certain area. Due to this specialization, this person is less capable in other areas.
- **The Generalist:** Someone who is at home within several areas but is never as good in a specific area then a specialist is.



*Figure 3: The “Jack-of-all-trades is a master of none” principle: light blue indicating the Specialist, dark blue indicating the Generalist.*

Before we can answer the question on a possible preference for generalist or for specialists, we should first look into the different areas of expertise for a test professional and talk about niches.

### ***The Tester’s Niche***

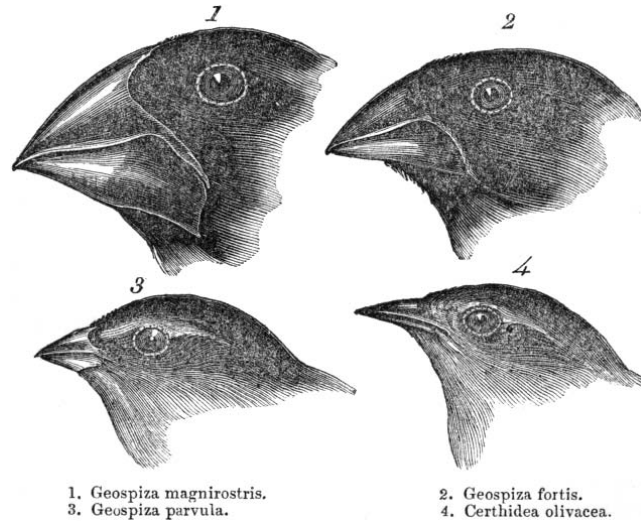
A niche is described as a place, employment, status, or activity for which a person or thing is best fitted (*Merriam-Webster*). The ecological niche describes how an organism or population responds to the distribution of resources and competitors (e. g., by growing when resources are abundant and threats are scarce) and how it in turn alters those same factors (e.g., limiting access to resources by other organisms, acting as a food source for predators and a consumer of prey).

What is the niche in which a testing professional belongs? As testing becomes more and more mature as a profession, more and more niches are discovered and adopted by testing professionals. Over the years testing has become much more than just checking whether an application works. Several niches have opened up during the last few years and new ones will open up in the future.

Testing has truly become a more professional IT discipline then ten years ago. Today, there is a growing need for:

- Testers with business knowledge
- Tooling
- Consultancy
- Niche Testers
- Development Testers

The key element to adaptation towards these niches is diversification. Darwin's finches are a classical biological example on diversification in order to adapt to new niches (Darwin, 1859). In order to survive the limited food sources present on the Galapagos Islands, a single bird species evolved through diversification in order to adapt to different niches that were present on the Galapagos Islands. This resulted in the existence of 13 new subspecies.



Finches of the Galapagos Archipelago

Figure 4: Diversification in Darwin's finches

Again, it is not very hard to project these principles on the testing industry. More and more, the testing domain is maturing, resulting in more and more niches that are becoming available.

Testing has truly become a more professional IT discipline then ten years ago. Today, there is a growing need for:

- **Testers with business knowledge:** Will we grow towards financial testers, utilities testers, life sciences testers?
- **Tooling:** Tooling and scripting wizards
- **Consultancy:** Conceptual testers who can do test assessments and write test methodologies.
- **Niche Testers:** Performance testers, security testers, usability testers, SAP testers, mainframe testers
- **Development Testers:** Testers who know what a unit test tool is and who also know how to use it to create automated unit tests (Rollinson 2008)

It is very possible that the testing profession will also be subdivided in several "sub-species". This is a trend that already underway.

*Where we as testing professionals, however, can make the difference for the customer is in providing a solid testing team with a good mix of specialists and generalists. This kind of team structure ensures that their combined effort will enable the customer to receive good product quality, as inexpensively and as quickly as possible.*

## The Driving Forces of a Tester's Evolution

### ***The Testing Professional: A Jack-Of-All-Trades Becomes a Master of All?***

Another result of the testing profession becoming more mature is the following:

- Testing is becoming a commodity. More and more companies are and will be occupying themselves with software testing. They will hire test professionals to staff them at certain projects. How can one test professional stand out among the others?

Looking at the different niches that have been identified above, one would think that becoming a specialist is the way of the future for the test professional.

It is important, however to consider the following question:

- Is the customer willing to pay for a testing professional who is a specialist within a certain area?

If we would let the customers express their preferences freely, their preference would be a testing professional who is a specialist on all areas. That is, a generalist with specialist knowledge in all areas. A jack-of-all-trades, but now a master of all.

### ***Human Restrictions and How to Survive The Future***

As the testing domain keeps on expanding towards the future, more and more niches will be uncovered. And with every niche that comes into existence, it will become harder and harder for a testing professional to remain up-to-date on all these different areas. After all, we are only humans and will always encounter limitations to our capacities.

Therefore, the customer's ultimate wish for a test professional who knows everything there is to know about testing shall remain utopian. A specialist may be an excellent fit for a large enough project where he can be occupied with his specialty full-time. Within smaller projects or companies, a generalist might be more suited.

Where we as testing professionals, however, can make the difference for the customer is in providing a solid testing team with a good mix of specialists and generalists. This kind of team structure ensures that their combined effort will enable the customer to receive good product quality, as inexpensively and as quickly as possible. Only then will we be able to overcome the testing paradox and ensure customer adoption of testing. Once the testing community is able to do this, they will be able to secure our future as test professionals. Or even better, they will be able to create a bright, interesting future for the testing profession with many interesting new challenges.

## References

- Aharonovitz, M. (2007), *Three Tips to Improve Your Requirements-Based Testing (RBT)*, Borland  
([http://www.borland.com/us/company/newsletter/issue5/3tips\\_improve\\_rbt.html](http://www.borland.com/us/company/newsletter/issue5/3tips_improve_rbt.html))
- Beck, K. (2002), *Test Driven Development: By Example*, Addison-Wesley Longman
- Binstock, A. (2008a), *Is the popularity of Unit Tests waning?*,  
(<http://binstock.blogspot.com/2008/05/is-popularity-of-unit-tests-waning.html>)
- Binstock, A. (2008b), *Is Unit testing Doomed?*,  
([http://www.techworld.com.au/article/256619/unit\\_testing\\_doomed](http://www.techworld.com.au/article/256619/unit_testing_doomed))
- Boehm, B.W. (1981), *Software Engineering Economics*, Englewood Cliffs, N.J. : Prentice-Hall
- Darwin, C. (1859), *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life* (1st ed.), London. John Murray
- Evans, D., Pointon, N. (2005), *Meet the New Tester; Professional Tester* (editions March 2005, June 2005)
- Gunderloy, M. (2004), *Coder to Developer: Tools and Strategies for Delivering your Software; chapter 5: Preventing Bugs with Unit testing*, Sybex
- IEEE 610 (1991), *IEEE Computer Dictionary - Compilation of IEEE Standard Computer Glossaries*, IEEE
- IEEE 1028 (1997), *IEEE Standard for Software Reviews*, IEEE
- IPL (1997), *Why Bother To Unit Test?* IPL
- ISTQB (2007), *Standard glossary of terms used in Software Testing*, ISTQB  
(<http://www.istqb.org/downloads/glossary-current.pdf>)
- Martin, J. (1984), *An Information System Manifesto*, Prentice Hall
- Merriam-Webster, *Definition of the word niche*  
(<http://www.merriam-webster.com/dictionary/niche>)
- Narayanan, S. (2002), *Defect Migration*  
(<http://www.sodhanai.com/articles/defectmigration.pdf>)
- Koch, G. (2008), *Unit testing Tools Unlikely To Transform Life Or Industry*, Software Test & Performance Magazine, January 2008
- Rollinson, B.J. (2008), *Break the Black Box Barrier*, Software Test & Performance Magazine, September 2008
- Savoia, A. (2005), *The Developer Testing Paradox*  
(<http://www.developertesting.com/archives/month200501/20050127-TheDeveloperTestingParadox.html>)
- Savoia, A. (2007), *Testing Genes, Test Infection, and the Future of Developer Testing*,  
(<http://www.developertesting.com/archives/month200701/20070126-developer%20testing%20test-infected%20gene.html>)
- Van Hese, Z. (2007), *Software Testing: A Profession of Paradoxes*, CTG
- Wikipedia, *On Natural Selection*  
(<http://en.wikipedia.org/>)
- Wikipedia, *On Darwin's finches*  
(<http://en.wikipedia.org/>)

***For more information  
about CTG's Testing  
Services, please contact:***

Bert Jagers  
Testing Consultant  
CTG Belgium NV  
Woluwelaan 140 a Bus 3  
1831 Diegem, Belgium

Tel: +32 2 714 00 24  
Fax: +32 2 725 09 20  
Email: [bert.jagers@ctg.com](mailto:bert.jagers@ctg.com)

*Backed by over 40 years' experience, CTG provides IT solutions and services to help our clients use technology as a competitive advantage to excel in their markets. CTG combines in-depth understanding of our clients' businesses with a full range of integrated offerings, best practices, and proprietary methodologies supported by an ISO 9001:2000-certified management system. Our IT professionals based in an international network of offices in North America and Europe have a proven track record of delivering high-value, industry-specific solutions. CTG serves companies in several industries and is a leading provider of IT and business consulting solutions to the healthcare market.*

*More information about CTG is available on the Web at [www.ctg.com](http://www.ctg.com).*

**A SPECIAL CLIENT REPORT**

---

*Confidential Information: Special Report and Analysis for CTG Client Use Only*