



Test-Driven Development A Commitment to Quality

When hiring a development firm, clients have a right to expect great software that works. Some firms are unable to satisfy this expectation because their approach to software development is not focused on quality. However, a firm that incorporates test-driven development (TDD) into its development process is able to prevent most software defects from being introduced in the first place. The result: clean code that works. This white paper describes test-driven development and explains why business leaders should consider it a must-have characteristic of any development firm.

Introduction

To business leaders, the software development process can often seem like a black box with mysterious inner workings. However, scrutiny of those inner workings—which are not so mysterious when you take away the jargon—can provide valuable insight when you need to hire a software development firm. Understanding how each firm builds its software is a prerequisite for evaluating its ability to deliver a great product, and a reliable way to differentiate between competing bids.

What should a client expect from a development firm? High-quality software that works. Unfortunately, not all development firms are able to satisfy this reasonable expectation.

Waterfall Development

Traditional software development takes a *waterfall* approach, meaning that software is developed in a linear sequence of discrete steps that include design, programming, and eventually, debugging. A serious weakness of waterfall development is that software interdependencies are frequently discovered and identified near the end of the process, during the debugging phase.

Conflicts that arise from these unforeseen dependencies are a common source of bugs. By the time testers go to work on the software, the number of bugs—more properly referred to as defects—can be large, and many are deeply woven into the fabric of the software, making them more difficult and costly to fix.

Similar to other industries, in software development, early changes are relatively cheap. Changes grow increasingly expensive as a project progresses because more work is required to achieve the same outcome.

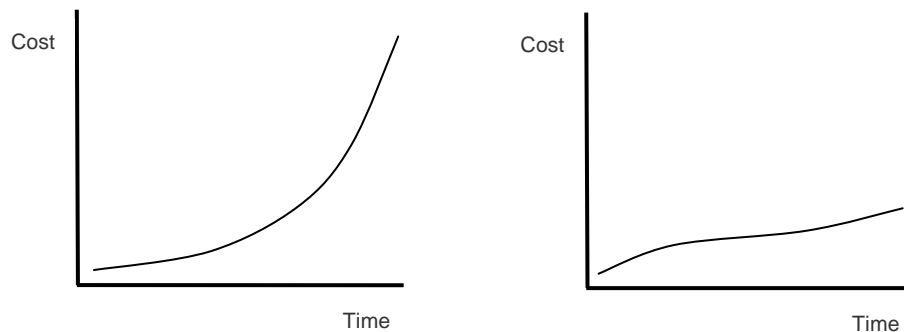


Figure 1: Cost of change using (i) waterfall, and (ii) Agile/TDD.³

According to some estimates, half of software development time is spent on finding and fixing defects.⁴ Project managers, however, tend to underestimate debugging time, so quality assurance activities regularly disrupt software project schedules. When time runs out, the software is often shipped full of defects that must be addressed in subsequent releases.

...for every example of a fatal bug there are a thousand instances of bugs that cause property loss or financial damage. Software problems have been blamed for blackouts, airline delays and airport closings, bank account foul-ups, auto recalls, and virtually every other kind of snafu and inconvenience you can imagine.

- Scott Rosenberg
Dreaming in Code¹

Sequential development leaves testing until the end of the development process—exactly the wrong time to find defects. There is no question that finding and fixing defects late in the development cycle is very expensive...In fact, people working in sequential processes expect to discover defects during final testing...

- Mary & Tom Poppendieck
Leading Lean Software Development²

Most projects are on schedule and within budget until testing starts, at which time excessive defects stretch out testing by several hundred percent compared with plans and cost estimates.

- Casper Jones
Software Engineering Best Practices⁵



TDD: A Lean Approach

On the other hand, what if a development firm took a more lean approach, by preventing defects from being introduced in the first place? Put another way, what if defects could be engineered out of the development process? Naturally, that would be much more efficient than having to find and fix them after the fact. Finding defects is waste but preventing defects is lean.

Consider the second curve in Figure 1, which shows the *cost of change* in an Agile approach that includes *test-driven development (TDD)*. The Agile/TDD curve is much flatter than the waterfall curve because the cost of finding and fixing defects remains practically constant through much of the development phase.

The reason for this low fixed cost is that, by integrating some aspects of quality assurance into the code writing process, programmers find and fix most defects immediately after creating them, instead of relying on testers to root them out during a separate debugging operation, late in development. If time is at a premium, programmers can simply flag major problems and add them to the agenda for the next code iteration.

When testing is an ongoing process, the result is clean code built on clean code and hence no domino effect with respect to bugs. By investing in TDD up front, a business can prevent technical debt from accumulating, which is what happens when waterfall methods are employed.

Business leaders who are choosing a development firm for their software project should understand that test-driven development is a key factor in the decision making process. **This white paper describes how TDD works and explains why it's the only the way that software should be built.**

Test Driven Development (TDD)

In test-driven development, programmers continually write short bits of code that test the software for specific outputs, thereby ensuring that it does what it's supposed to do. After writing each test, the programmer then adds just enough code to the software to pass the test. By forcing each small unit of code to meet the technical criteria defined by the business requirements, the TDD process automatically filters out most defects at the source.

Even better, the tests remain embedded in the code, forming a permanent test suite that can be used to check the integrity of the software throughout its lifecycle—development, maintenance, and new releases.

With its focus on incremental development of clean code, TDD fits well with Agile principles. Consider these three statements from the Agile manifesto:⁸

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Working software is the primary measure of progress.
3. Simplicity—the art of maximizing the amount of work not done—is essential.

We should know that it is best to avoid defects in the first place, or at least to find defects immediately after they have been inserted into the code. It's the same as losing your keys: If you lost your keys a few seconds ago, look down. If you lost them a few minutes ago, retrace your steps. If you lost them a week ago, change the locks. And yet we wait until the end of the development process to test our code, and by that time we need a lot of locksmiths.

- Mary & Tom Poppendieck
Leading Lean Software
Development⁶

...[Google] estimated that a bug found during TDD costs \$5 to fix, which surges to \$50 for tests during a full build and \$500 during an integration test. It goes to \$5000 during a system test. Fixing bugs earlier would save them an estimated \$160M per year.

- Gojko Adzic
Improving Testing Practices
at Google⁷

No software engineers release even the tiniest change without testing, except the very confident and the very sloppy.

- Kent Beck
Test-Driven Development
By Example⁹



Programming is demanding. It requires perfection, consistently, for months and years of effort...People aren't so good at perfection. No wonder, then, that software is buggy. Wouldn't it be cool if there was a tool that alerted you to programming mistakes moments after you made them—a tool so powerful that it virtually eliminated the need for debugging? There is such a tool, or rather, a technique. It's test-driven development, and it actually delivers these results.

- Shore & Warden
The Art of Agile
Development¹⁰

A hallmark of good test-driven development is testing that can be read as documentation of the production code.

- Amr Elssamadisy
Agile Adoption Patterns¹¹

Test-driven development helps programmers write simple, clean code that works, making it easier for Agile teams to efficiently and continually deliver valuable software to their clients.

TDD also eliminates waste from the development process and is therefore a key component of *lean software development*, as well.

TDD promotes lean principles by:

- Bringing clarity to software requirements.
- Encouraging automated testing.
- Eliminating defects at the source, thereby minimizing the impact of quality assurance activities near the end of the development process.
- Continually delivering working software in small increments so the client doesn't have to wait.
- Allowing testers to work to their strengths, unburdened by unrealistic expectations and deadlines.

With TDD, programmers continuously alternate between developing and testing. They fix problems as they go rather than making changes months later when the code is no longer fresh in their minds. They simultaneously extend the software and scrub it clean. Because of this, test-driven development offers programmers a critical path to their objective, free of unnecessary detours (like tracing the root cause of elusive defects) and wasteful U-turns (like rewriting code).

An added bonus of TDD is that all stakeholders—developers, management, and clients—have more confidence in the software because they know that the test suite, consisting of the programmers' many small tests, will detect problems immediately after a change is made to the code.

User Stories

Business requirements—what the client wants the software to do—are often expressed in short statements called *user stories*. A user story describes one capability or behaviour that the software must have in order to satisfy a typical user. For example, a user story might read, “As a typical business user, I need to see a list of all my active accounts immediately after I log in to the system.”

When clients work with the development team to write clear, concise user stories, they also formulate *acceptance tests* as a way of verifying that the software will satisfy their user stories. An acceptance test for the previous user story example would confirm that each user's list of active accounts is displayed after login.

With user stories and acceptance tests, clients have all the tools they need to confidently steer software development in a direction that satisfies their evolving business requirements. In the end, they get software with the right features.



Typical Steps

In an Agile sprint (iteration), the development team usually commits to writing code for a list of user stories.

Using TDD, here's what they do to produce code for each user story:

1. Write a test

When the sprint is initiated, programmers write *unit tests* that check the code for specific outcomes defined in the user story. It's important to limit the scope of each test so that a small chunk of code is sufficient to pass it. A programmer may have to write numerous unit tests and code segments to satisfy a single user story.

...TDD cranks up the feedback on the execution of your code. Every few minutes—as often as every 20 or 30 seconds—TDD verifies that the code does what you think it should do. If something goes wrong, there are only a few lines of code to check. Mistakes are easy to find and fix.

- Shore & Warden
The Art of Agile Development¹²

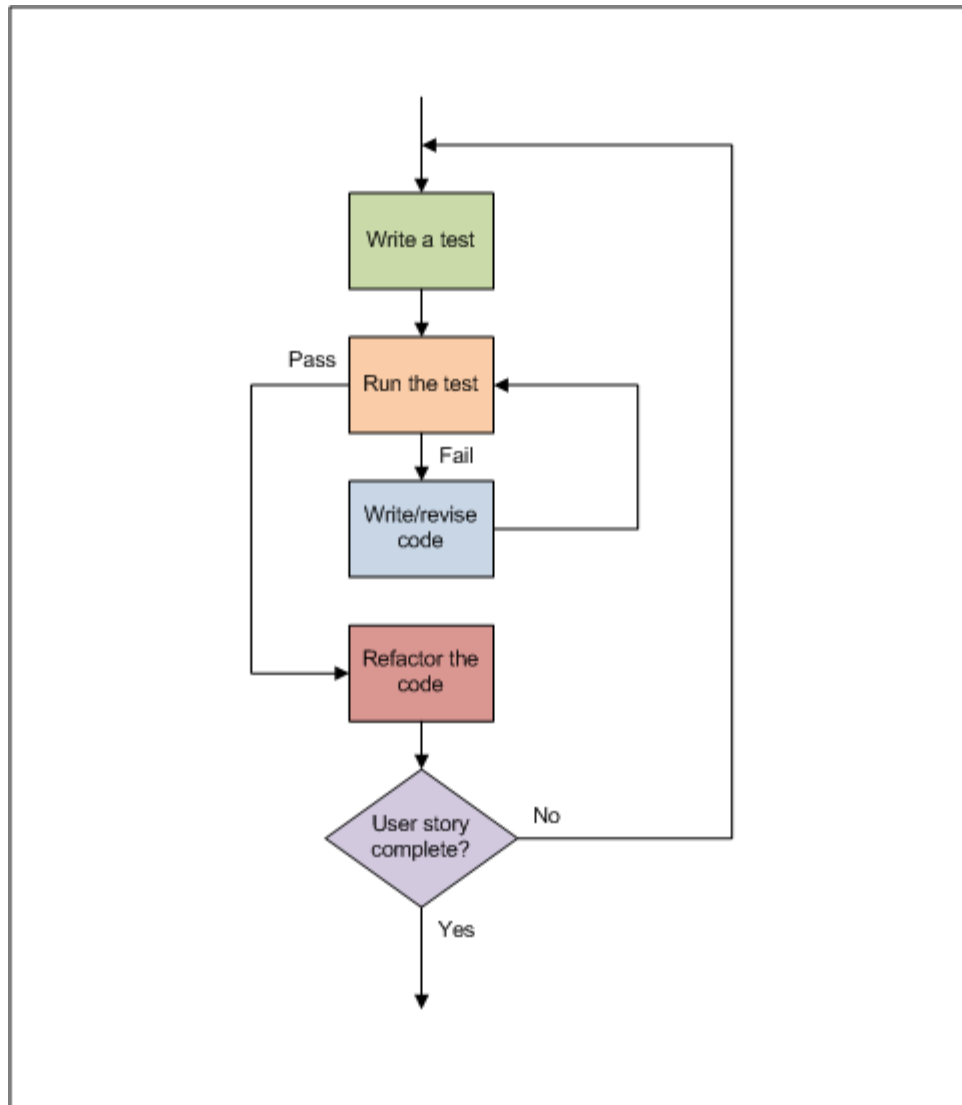


Figure 2: Test-driven development—basic process



Test-driven development provides a natural rhythm with its think-test-design-refactor cycle. The successful end of every cycle produces a stable checkpoint at which the entire system works as designed; it's a solid foothold from which to continue. If you go wrong, you can quickly revert to the prior known-good state.

- Shore & Warden
The Art of Agile Development¹³

...TDD should only form part of an integrated approach to testing...TDD is testing by programmers, for programmers, while other tests are more likely to require the attention of testing specialists.

- Steve Hayes
Test Driven Development Explained¹⁴

After TDD is finished, the tests remain. They're checked in with the rest of the code, and they act as living documentation of the code. More importantly, programmers run all the tests with (nearly) every build...If someone accidentally changes the code's behaviour...the tests fail, signalling the mistake.

- Shore & Warden
The Art of Agile Development¹⁵

Again, to be clear, programmers use unit tests to check small chunks of code while clients (business people) use acceptance tests to ensure that specific business needs are met by the software. As a client adds more user stories, programmers can confidently write new code, knowing that conflicts with existing code (for other user stories) will be immediately exposed by the failure of one or more tests.

One noteworthy benefit of TDD is that the suite of unit tests becomes an executable specification of requirements, a traceability feature that's very handy for regulators and other stakeholders who must review the code.

2. Run the test

After writing a unit test, the programmer checks the test by running it to ensure that it fails. Written properly, a test should always fail until the programmer writes code that will satisfy it.

3. Write some code

The programmer writes just enough code to pass the test. Simple, unambiguous code makes it easy to diagnose and fix problems as they arise.

4. Refactor the code

After writing a test and a piece of code to pass it, the programmer then *refactors*, or optimizes, the code. Programmers use refactoring tools to clean up their code—making it more efficient—without altering its functionality.

Refactoring can cause waves of defects throughout the code, so the unit tests are used to detect what's been inadvertently broken. For this reason, refactoring is typically done in small increments so programmers can quickly zero in on problem areas and fix them.

QA Unleashed

Using TDD, programmers tend to write error-free code, but no system is perfect and some defects will inevitably creep into the software. However, development teams that use TDD are able to limit defects to a small number, which allows professional software testers to concentrate their efforts on activities that add real value, like:

- **Integration testing**
Seeing how individual units of software interact.
- **Exploratory testing**
Checking the software's operational boundaries.
- **Load testing**
Checking performance, scalability and other attributes.
- **Security testing**
Looking for openings that attackers can exploit.

TDD frees up testers to do what they're good at, which includes identifying weaknesses or conflicts in the requirements, and suggesting timely



Technical debt builds up when the development team takes shortcuts, hacks in quick fixes, or skips writing or automating tests because it's under the gun. The code base gets harder and harder to maintain. Like financial debt, "interest" compounds in the form of higher maintenance costs and lower team velocity.

- Crispin & Gregory
Agile Testing¹⁶

...test-driven development significantly increases quality to market, time to market, and product lifetime. It also increases flexibility and reduces the cost of development...

- Amr Elssamadisy
Agile Adoption Patterns¹⁷

Test-driven development (TDD) really shook the world of software development from the very foundations...Ten years ago it may have been some sort of mystical programming skill only known to true kung-fu masters. Today, it is becoming common sense...Whether test-driven development should or should not be done is no longer an issue...

- Gojko Adzic
Test-Driven .NET
Development with
FitNesse¹⁸

improvements. The result is better working code at every stage of development—without the burdensome expectation of finding numerous defects late in the process.

Advantages of TDD

The main advantages of test-driven development are:

- **Technology investment, not technology debt**
With software built on clean code, there aren't any lingering defects that have been deferred to a later date.
- **Software that's cheaper and easier to maintain**
Code is shorter and has fewer defects, requiring less effort to maintain. Maintenance is typically the most costly aspect of software—usually much more expensive than development.
- **Valuable working code delivered on a continual basis**
Each piece of code is clean, functional, and directly related to a user story under development in the current sprint. At the end of each sprint, the list of user stories has been developed into working software.
- **No frenzied, last-minute quality assurance effort**
Programmers find and fix most defects immediately after introducing them. Major debugging at the end of the development process is expensive, time consuming, and damaging to project schedules.

Questions to Ask

How can you tell if a development firm is likely to build software that's free of defects? Quite simply: Scrutinize its development process.

Here are some questions that you should ask a development firm before signing on the dotted line:

- How do you prevent defects in your software?
- How do you detect any that are present?
- How do you know you haven't missed any?
- What software engineering best practices do you use?
TDD?
- At what stage of development do you find and fix defects?
- How do you define "high-quality" software?



References

1. Rosenberg, Scott. *Dreaming in Code: Two Dozen Programmers, Three Years, 4732 Bugs, and One Quest for Transcendent Software*, (New York: Random House, 2008), 248.
2. Poppendieck, Mary and Tom. *Leading Lean Software Development: Results are Not the Point*, (Upper Saddle River, New Jersey: Addison-Wesley, 2010), 70.
3. Ambler, Scott W. "Examining the Agile Cost of Change Curve", agilemodeling.com, accessed May 6, 2010.
4. Zeller, Andreas. *Why Programs Fail: A Guide to Systematic Debugging (Second Edition)*, (Burlington, Massachusetts: Elsevier, 2009), xv.
5. Jones, Casper. *Software Engineering Best Practices: Lessons from Successful Projects in the Top Companies*, (New York: McGraw-Hill, 2010), 518.
6. Poppendieck, *Leading Lean Software Development*, 71.
7. Adzic, Gojko. "Improving Testing Practices at Google", gojko.net, December 7, 2009.
8. "Principles behind the Agile Manifesto", <http://agilemanifesto.org/principles.html>, accessed May 6, 2010.
9. Beck, Kent. *Test-Driven Development By Example*, (Boston: Pearson Education, 2003), 123.
10. Shore, James, and Shane Warden. *The Art of Agile Development*, (Sebastopol, California: O'Reilly Media, 2008), 289.
11. Elssamadisy, Amr. *Agile Adoption Patterns: A Roadmap to Organizational Success*, (Boston: Pearson Education, 2009), 281.
12. Shore, *The Art of Agile Development*, 290.
13. Shore, *The Art of Agile Development*, 374.
14. Hayes, Steve. "Test Driven Development Explained", builderau.com.au, August 27, 2004.
15. Shore, *The Art of Agile Development*, 290.
16. Crispin, Lisa, and Janet Gregory, *Agile Testing: A Practical Guide for Testers and Agile Teams*, (Boston: Pearson Education, 2009), 106.
17. Elssamadisy, *Agile Adoption Patterns*, 277.
18. Adzic, Gojko. *Test-Driven .NET Development with FitNesse*, (London: Neuri, 2008), 3.





Architech Solutions is a Toronto-based technology consulting and software development firm. We design and build powerful, user-centred systems that work. We're agile, disciplined, and passionate about delivering for our clients.

Architech Solutions
3 Church Street
Suite 602
Toronto, Ontario
M5E 1M2

Phone: (416) 607-5618
Fax: (416) 352-1768

To book a free on-site Discovery Workshop led by our team of consultants, e-mail info@architech.ca or visit us at www.architech.ca

