What are the limits of TDD? [closed]

Asked 15 years, 8 months ago Modified 15 years, 8 months ago

Viewed 1k times

8

Closed. This question is opinion-based. It is not currently accepting answers.

Want to improve this question? Update the question so it can be answered with facts and citations by editing this post.

Closed 11 years ago.

Improve this question

I have just recently discovered how much I enjoy developing code the TDD way: I feel I have much more control over the direction the development is going. Whereas before I spent a lot of time designing data structures and algorithms up front, now I start small and "grow" my code organically. After every red/green/refactor cycle I have code that does *something*. It feels like my code is a living thing and I am directing where it should grow. I don't know if that's how everyone feels when they get into TDD, but this is my experience. And it strikes me that this is so similar to how successful free software projects are grown rather than designed.

However, now that I have got a hang on the test-driven development, I am beginning to wonder what its limits are. It seems to be quite useful for developing functional code: feed this input to that function, and you get this result. But this is just a small part of what software development is about. What about GUI development, networking, database development, web applications? What is your experience? Have you ever tried TDD with any of these types of development? Do you know of any tools or frameworks? Can you recommend any articles or books?

tdd

Share

Improve this question

Follow

asked Apr 23, 2009 at 13:40

ajanicij

415 ● 3 ● 12

5 Answers

Sorted by: Highest score (default)

9

"What about GUI development, networking, database development, web applications?"

Why wouldn't it work?

GUI. The only thing TDD can't do well is evaluate the "look" of the interface. But it can evaluate the behavior. If you did your design well (separating model, view and control), you can test the control and model easily as

TDD. view, however, is more difficult to write tests for. ("assert that the button is below the field" isn't sensible.)

Networking. Not sure what this means. However, defining RESTful web services works out very nicely when done via TDD. Define the URI's, write test cases, and then build the services that provide expected responses.

Web Applications. The Django web framework directly supports TDD for development. They have unit tests for the web data model and control layers. Plus, they have unit tests for the HTML page presentation and behavior.

Share Improve this answer Follow

answered Apr 23, 2009 at 13:58

S.Lott
391k ● 82 ● 517 ● 788

9

I am wondering what framework/language you use for your TDD if you have not yet hit the most obvious parts of where it is difficult:

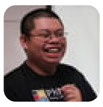
- **GUIs** - Certain environments and platforms, specifically, Windows Forms and ASP.NET web forms, suffer from practical untestability, because there is no easy way to isolate or mock their behavior. This is one of the main driving forces of ASP.NET MVC, for that matter.
- **Persistence** - Any sort of persistence, whether it be disk storage, connection to a database, to a network, or some other form of external system would be a

limitation, as all of them are persistence tests which depend too much on the state of an external component to succeed. These integration tests then are considered *fragile tests*. Mocking is used to mitigate the impact of such external systems.

- **Adoption** - Even if you are well versed in TDD, it's an uphill battle trying to get other developers -- much less whole development shops -- to use it. Many just don't see the benefit; they are easily turned off by the initial steep learning and productivity curves. You will encounter cases wherein you're the only one practicing TDD in your project and even if you enforce it the other developers will introduce low-quality, senseless tests. This non-technical reason is the biggest hurdle to using TDD in real-life production systems that I have faced, and it's a painful realization when you're quite aware of the benefits.

Share Improve this answer Follow

answered Apr 23, 2009 at 14:08



Jon Limjap
95.3k ● 15 ● 103 ● 153

ooh +1 for persistence - that's such a pain no matter how well mocked – annakata Apr 23, 2009 at 14:11

2 +1 for adoption. From my point of view it's the major "issue" with TDD (most people simply don't see the benefit or give up TDD after first trying). – Anderson Matos Oct 7, 2011 at 3:29

You have some costs to doing TDD:

3

- When you update your code you must maintain your unit tests to check for the new correct behavior.
  - Once you have a large amount of unit tests, when you refactor, you need to rewrite the unit tests

I like TDD for lots of things. Keep in mind that the above maintenance cost of keeping working unit tests might mean that although your code has become more modular, the chosen modular organization is harder to change. You can't reorganize your code without the additional cost of rewriting all your unit tests. That's really the big limit I've found with TDD. Later when you realize you've made a mistake, now you have all this overhead that you have to go through to refactor things whereas before these changes could happen much more fluidly. So to use TDD well I'd say you need to make good decisions when first developing the module. Not always easy in the changing, agile world we live in ;).

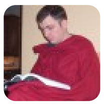
Also consider:

- Not everything can be purely black-box tested through a classes methods. IE a method that's result is dependent on exactly how many microseconds have passed since it was last called. You need to add hooks to control how much time has passed and see if you get the correct response.
- Unit testing is really not appropriate for everything. Seeing if a GUI is usable by humans can really never be proven via unittest.

Share Improve this answer Follow

edited Apr 23, 2009 at 14:18

answered Apr 23, 2009 at 13:55



Doug T.

65.5k ● 28 ● 141 ● 205

Why do you mention "black-box tested"? What's that got to do with TDD? – S.Lott Apr 23, 2009 at 13:59

I don't see why a method whose result is time-dependent could not be tested: in your test you can call the method, then wait, then call it again. My problem is with external factors, such as the network or humans, which can't be controlled by the test code. – ajanicij Apr 23, 2009 at 14:09

3 Rewriting tests is really part of the mantra of TDD. It's useless to go TDD if you are unwilling to accept the fact of life that software is very malleable and can change in so many ways. – Jon Limjap Apr 23, 2009 at 14:13

@ajanicij -- well if its a real time method that is dependent on calculating something precisely on how many microseconds has passed, you can't really just sleep 500 microseconds. @Jon Limjap - that doesn't mean that it's any less of a cost to always have to rewrite your unit tests to respond to these changes... – Doug T. Apr 23, 2009 at 14:15

@S.Lott -- I mean black box with respect to testing the class purely through its public methods. Not black box with respect to the end application. – Doug T. Apr 23, 2009 at 14:16

As far as limits go it's important to remember that TDD is about stability and managing change - arguably it doesn't

2

particularly help you design *better* it just enforces the implementation of your design*. Actually coming up with good system architecture still requires as much thought as with any other dev methodology.

A variety of limits exist, most of which are because software still requires humans:

- TDD does nothing to help intangibles like GUI layout and general user experience
- TDD does not solve garbage-in garbage-out in your design
- TDD does not solve environmental problems (i.e. networking, I18N and time-dependant bugs tend to slip through cracks)
- Unit-tests themselves need design (solved to a greater or lesser extent depending on framework)
- Unit-tests can increase the amount of work required when refactoring (but hopefully this has been minimised by your beautiful design).
- And the classic: singletons are hard->impossible to unit-test (the general solution here is to avoid singletons)

* Even as I've written that, I'm unsure how to phrase that correctly.

Share Improve this answer Follow

answered Apr 23, 2009 at 14:06

annakata
75.7k ● 18 ● 115 ● 180

As I see TDD, it changes the nature of design: you grow your design with the code. Q: why are singletons hard to unit-test? – ajanicij Apr 23, 2009 at 14:13

1 Singletons are not necessarily hard to unit test. Code that uses them can become hard to unit test. – Yorgos Pagles Apr 23, 2009 at 14:18

1 @YoP - better said. Singletons don't play well with unit-testing for a variety of reasons depending on language, for example they typically manage their own construction and maintain a singleton state. It's hard to know if that state will effect later tests or not. It's hard to know if the singleton will be created/destroyed within the cycle of an individual test. Etc.. Most problems are fixable, it's just a pain. – annakata Apr 23, 2009 at 14:34

-2

Try groovy on grails. You get automatically generated tests.

Every time you create a controller class automatically a unit test class has been generated.

From the book "The definitive guide to grails":

Grails separates tests into “unit” and “integration” tests. Integration tests bootstrap the whole environment including the database and hence tend to run more slowly. In addition, integration tests are typically designed to test the interaction among a number of classes and *therefore require a more complete application before you can run them. Unit tests, on the other hand, are fast-running tests, but they require you to make extensive use of mocks and stubs. Stubs are classes used in testing that mimic the real behavior of methods by returning arbitrary hard-coded values. Mocks essentially do the same thing, but exhibit a bit more intelligence by having "expectations." For example, a mock can specify that it "expects" a given method to be invoked at least once, or even ten times if required.*

Share Improve this answer Follow

answered Apr 23, 2009 at 13:52

Luixv
8,710 ● 21 ● 86 ● 122

---

4 Automatically generating tests is NOT the point of TDD. In fact, once tests are autogenerated, the point is lost. TDD is a design tool - not a testing tool. – Jon Limjap Apr 23, 2009 at 13:53

-1 due to misunderstanding the TDD concept – Nathan W Apr 23, 2009 at 13:59

TDD is test Driven Design, isn't it? Grails provides an environment which suites for TDD. I.e. makes your life easier. – Luixv Apr 23, 2009 at 14:09

1 With TDD you are creating your tests before you write any code, thus having a tool auto generate tests from existing code defeats the intent of TDD. And TDD stands for Test Driven Development. – Nathan W Apr 24, 2009 at 8:40