Is it feasible to introduce Test Driven Development (TDD) in a mature project? [closed]

Asked 16 years, 3 months ago Modified 12 years, 4 months ago

Viewed 4k times

37

As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, visit the help center for guidance.

Closed 12 years ago.

- Say we have realized a value of TDD too late. Project is already matured, good deal of customers started using it.
- Say automated testing used are mostly functional/system testing and there is a good deal of automated GUI testing.
- Say we have new feature requests, and new bug reports (!). So good deal of development still goes on.

- Note there would already be plenty of business object with no or little unit testing.
- Too much collaboration/relationships between them, which again is tested only through higher level functional/system testing. No integration testing per se.
- Big databases in place with plenty of tables, views, etc. Just to instantiate a single business object there already goes good deal of database round trips.

How can we introduce TDD at this stage?

Mocking seems to be the way to go. But the amount of mocking we need to do here seems like too much. Sounds like elaborate infrastructure needs to be developed for the mocking system working for existing stuff (BO, databases, etc.).

Does that mean TDD is a suitable methodology only when starting from scratch? I am interested to hear about the feasible strategies to introduce TDD in an already mature product.

unit-testing mocking tdd

Share

Improve this question

Follow

edited Aug 7, 2012 at 18:05

ThinkingStiff

65.3k ● 30 ● 147 ● 241

asked Sep 20, 2008 at 11:14

rpattabi
10.2k ● 5 ● 48 ● 55

6 Answers

Sorted by: Highest score (default)

27

Creating a complex mocking infrastructure will probably just hide the problems in your code. I would recommend that you start with integration tests, with a test database, around the areas of the code base that you plan to change. Once you have enough tests to ensure that you won't break anything if you make a change, you can start to refactor the code to make it more testable.

Se also Michael Feathers excellent book Working effectively with legacy code, its a must read for anyone thinking of introducing TDD into a legacy code base.

Share Improve this answer Follow

edited Sep 20, 2008 at 11:28

answered Sep 20, 2008 at 11:21

Akselsson
790 ● 4 ● 6

Thanks for the book suggestion. It looks like it is what I looked for. – rpattabi Sep 20, 2008 at 11:41

I think its completely feasible to introduce TDD into an existing application, in fact I have recently done it myself.

16

It is easiest to code new functionality in a TDD way and restructuring the existing code to accommodate this. This way you start of with a small section of your code tested but the effects start to spread through the whole code base.

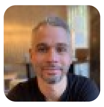
If you've got a bug, then write a unit test to reproduce it, refactoring the code as necessary (unless the effort is really not worth it).

Personally, I don't think there's any need to go crazy and try and retrofit tests into the existing system as that can be very tedious without a great amount of benefit.

In summary, start small and your project will become more and more test infected.

Share Improve this answer Follow

answered Sep 20, 2008 at 11:22



Garry Shutler
32.7k ● 13 ● 89 ● 120

Writing new unit tests around bugs works well. You don't have a "complete" test suite, but you have something to build on. – S.Lott Sep 20, 2008 at 20:31

9

Yes you can. From your description the project is in a good shape - solid amount of functional tests automation is a way to go! In some aspects its even more useful than unit testing. Remember that TDD != unit testing, it's all about short iterations and solid acceptance criteria.

Please remember that having an existing and accepted project actually makes testing easier - working application is the best requirements specification. So you're in a better position than someone who just have a scrap of paper to work with.

Just start working on your new requirements/bug fixes with an TDD. Remember that there will be an overhead associated with switching the methodology (make sure your clients are aware of this!) and probably expect a good deal of reluctance from the team members who are used to the 'good old ways'.

Don't touch the old things unless you need to. If you will have an enhancement request which will affect existing stuff then factor in extra time for doing the extra set-up things.

Personally I don't see much value in introducing a complex infrastructure for mock-ups - surely there is a way to achieve the same results in a lightweight mode but it obviously depends on your circumstances

Share Improve this answer Follow

edited Jun 7, 2011 at 12:28

johnsyweb
141k ● 26 ● 194 ● 251

answered Sep 20, 2008 at 11:23

Ilya Kochetov
18.4k ● 6 ● 47 ● 62

1 +1 for "TTD != unit testing", I shall now fix that typo.
– johnsyweb Jun 7, 2011 at 12:28

5

One tool that can help you testing legacy code (assuming you can't\won't have the time to refactor it, is Typemock Isolator: Typemock.com It allows injecting dependencies into existing code without needing to extract interfaces and such because it does not use standard reflection techniques (dynamic proxy etc..) but uses the profiler APIs instead. It's been used to test apps that rely on sharepoint, HTTPContext and other problematic areas. I recommend you take a look. (I work as a dev in that company, but it is the only tool that does not force you to refactor existing legacy code, saving you time and money) I would also highly recommend "Working effectively with legacy code" for more techniques.

Roy

Share Improve this answer Follow

answered Sep 22, 2008 at 1:32

RoyOsherove
3,269 ● 2 ● 31 ● 26

3

Yes you can. Don't do it all at once, but introduce just what you need to test a module whenever you touch it.

You can also start with more high level acceptance tests and work your way down from there (take a look at Fitnesse for this).

Share Improve this answer

Follow

answered Sep 20, 2008 at 11:19

Maximilian

4,828 ● 6 ● 30 ● 28

2

I would start with some basic integration tests. This will get buy-in from the rest of the staff. Then start to separate the parts of your code which have dependencies. Work towards using Dependency Injection as it will make your code much more testable. Treat bugs as an opportunity to write testable code.

Share Improve this answer

Follow

answered Sep 20, 2008 at 13:13

David P

3,624 ● 3 ● 39 ● 54