The Influence of Test-Driven Development on Design of Object-Oriented Systems: A Qualitative Study with Developers in Industry

(Work in progress)

Mauricio Finavaro Aniche, Marco Aurélio Gerosa
Department of Computer Science, University of São Paulo - Brazil

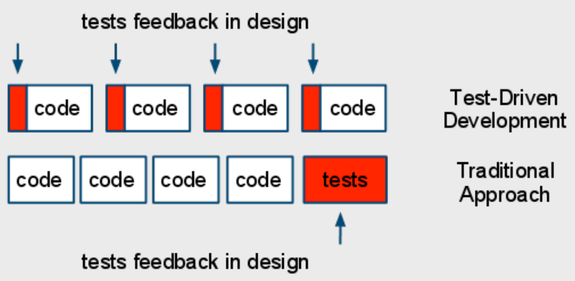
TDD

Focused on feedback. Test-Driven Development (TDD), popularized by Kent Beck through his book Test-Driven Development: By Example, is one of the agile practices focused on feedback. TDD has grate importante in the software development cycle since, as suggested by agile practices, the design should emerge as the software grows. And, in order to quickly respond to these changes, a constant feedback about the internal and external quality of the code is a need.

Becoming popular. TDD's popularity keeps growing among developers. The mechanic of the practice is simple: the developer should write a test before the implementation. In a more detailed explanation, the developer first should write a test that fails. Then, s/he make the test pass by implementing the desired feature. Finally, the developer should refactor the code. This cycle is also known as the "Red-Green-Refactor cycle", as it remembers the colors that a developer usually sees when doing TDD: the red means that the test fails, and the green means that the test has passed.

Design activity. Although TDD may look like a testing practice (after all, TDD has the word ``test'' in its name), its main contribution is the feedback in the class design. Most well-known authors, such as Kent Beck, Robert Martin, and Dave Astels, claim that this simple change in the traditional development cycle makes the code simpler, increases the class cohesion, and reduces the class coupling. Ward Cunningham, one of the Extreme Programming pioneers, affirm that "Test-First programming is not a testing technique"; a line that summarizes the main focus of the practice.

Earlier feedback. Traditional approaches have testing only after the feature is completely implemented and, consequently, they do not have the tests feedback during the classes initial design. The Figure below illustrates two programmers using different approaches, writing small pieces of code for a feature. TDDers constantly validate the design through tests, what does not happen in traditional approaches where many small pieces of code are written before writing the tests. An advantage of writing the tests before is to enable developers to take most part of design decisions while the cost of changes is still low.



Threats to Validity

Social Desirability Bias. Agile methods and TDD have a strong speech. Brazilian agile community is still young, and participants may respond what the community wants to hear. In this research, the participant may talk about what the literature says about TDD, and not about his real feelings about the practice.

Change of Attitude During Observation: During the observation, as they know they are observed, participants may change their behavior, and be more careful than usual to TDD.

Effects on Design

Test is the first client. The test if the first client of the class that the programmer is still about to write, and it makes him to think better about the behavior he expects from that class. Besides, programmers also think and decide about the class interface, such as class' and method's name, return types, and exceptions thrown. In addition, the programmer is encouraged to write a code that is easily testable.

The bigger the difficulty in writing a test, the bigger the possibility of some design problem. According to Michael Feathers, there is a synergy between a highly-testable class and a good design.

Effects on coupling and cohesion. TDD encourages developers to write loosely coupled components, in a way that they can be tested in an isolated way and, in a higher level, combined with other components.

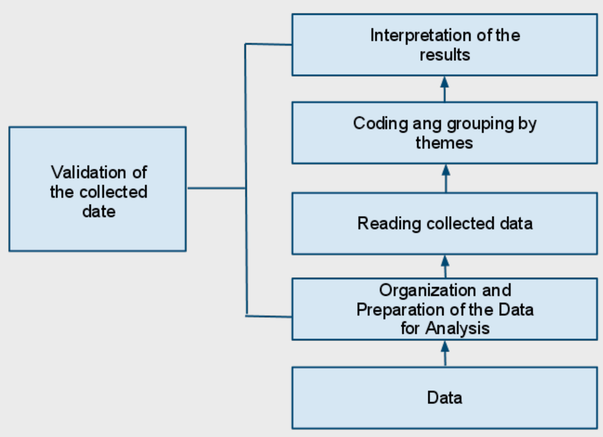
Experiment Planning

Qualitative study. Conducting an experimental study in software engineering has always been hard. One of the reasons for that is the human factor, highly present in software development process, as suggested by agile methods. Because of that, the analytic research paradigm is not enough to investigate complex real cases, involving people and their interactions with technology.

How TDD influences design decisions. This work does not try to infer a possible cause relation between TDD and good design, but to understand how the practice influences the design decisions taken by a TDD practitioner. In order to understand it, this study proposes a qualitative study with real industry TDD practitioners. Interviews and observations will be used as collect procedures.

Participants criteria. To conduct the study, some developers were selected given a few criteria: (i) practice TDD for more than 1 year; (ii) work with software development for more than 3 years; (iii) work on web projects that use Java; (iv) practice any agile software development method.

Validation. To validate this study, the following procedures will be adopted: The researcher will check all transcripts to make sure that no obvious mistakes happened; An additional researcher will be invited to review all generated codes; The results will be presented to the participants and their feedback will be collected; All possible bias will be discussed.



Interview Protocol

Open-ended questions. In addition, all question will be opened, allowing the developer to give a complete response about the subject.

Goals. *(i)* Define the programmers' experience on software development, good development practices, and TDD; *(ii)* The relation between the programmer's experience and the effects of TDD on design quality; *(iii)* The influence of another agile practices, such as pair programming, combined with TDD; *(iv)* The developer's understanding of TDD; *(v)* The effects of TDD on class coupling and cohesion; *(vi)* How the developers uses the test feedback to drive the design; *(vii)* The effects of TDD on simplicity.

Triangulation. Questions will try to triangulate the possible answers of the participant. The following model exemplifies how questions were formulated:
Do you usually see classes with many responsibilities?

(A)In your opinion, why do they appear?
(B)How do you deal with this problem?
(C)Does TDD influence on it?
(D)Do you notice any difference on cohesion when practicing TDD and when you don't practice TDD?
(E)Why do you think this difference is because of TDD? Can't it be because of your experience on software development?
(F)If it is the test that influences, what's the difference between writing the tests before or after?
(G)Thinking in code, what are the changes you make in the implementation, influenced by the tests?
(H)Do you think that a developer can create low coupled classes, even with TDD?

Observation

Protocol. The researcher will observe the participants during their daily activities. The minimum observation period will be 2 hours. After the period, the research will decide if the observation time should be prolonged. The minimum number of observations will be 10 per team, ensuring that each team will be observed for at least 20 hours.

Focus. Discussions about any topic below, but not limited by them, will be captured by the researcher:

(A)The influence of the test being the first client of a class;
(B)The explicit declaration of dependencies required by the unit test;
(C)Tests that show the lack of cohesion in a class;
(D)Tests that show bad coupling;
(E)Tests influencing on the simplicity;
(F)The usage of class composition techniques;
(G)The relation between the difficulty in writing an unit test and more flexible designs;

Contact Information

Mauricio Aniche
aniche@ime.usp.br

Marco Aurélio Gerosa
gerosa@ime.usp.br