DET - Developer's Exploratory Testing

Background

After talking about DET at several conferences, holding a number of workshops teaching DET and introducing it in many projects, I felt that now would be a good time to write an article about DET.

People used to say that good ideas are born from needs. In my first agile project for a couple of years, I was a test leader in a scrum team (yes, I know that such a role is not defined in Scrum) with no other dedicated testers. All team members were developers. For me it was obvious that in order to deliver good quality we all needed to do testing and do it as effectively and efficiently as we could. I just needed to convince others…

Test automation and unit testing are pretty axiomatic for developers but it is not the case with manual testing. I needed some manual testing method/process that could be easily explained and, at same time, effective in finding bugs.
I wrote a list of requirements to start with:

- **Everybody will do testing and contribute to it.**
  All team members are responsible for the quality of our product. This implies that all team members are responsible for testing to ensure good quality.
- **Developers will not test features they have developed by themselves alone.**
  While you are developing some feature you, of course, do testing but it is not enough. It should also be tested by someone who was not involved in development of the feature, hopefully with a more objective and critical attitude to the given feature.
- **We will not have comprehensive test specifications.**
  We should have only as much documentation as is needed which often depends on project context and customer demands.
- **The team agrees about found issues.**
  A big problem in many companies is the so called "ping-pong effect". All communication between developers and testers is done via some issue reporting system. The tester rises an

issue, the developer rejects it; the tester reopens it again and so on. It is a waste of time and can cost a lot of money. By making the team agrees on issues we avoid that and eliminate a lot of waste.

- **The process could be easily explained.**
  Having Scrum as the ideal, I wanted the test process to be similar regarding how easy it is to explain. The easier it is to explain the easier it is to accept it.
- **The communication and interaction between team members is vital.**
  Learning by doing and learning from mistakes is the most powerful way for team members to improve their effective testing.

DET's Characteristics

After putting everything together, ***Developer's Exploratory Testing*** was born. The word '***Developer'*** in the name is mainly to point out that this method is customized to fit developer teams.

It is very similar to Sprint:

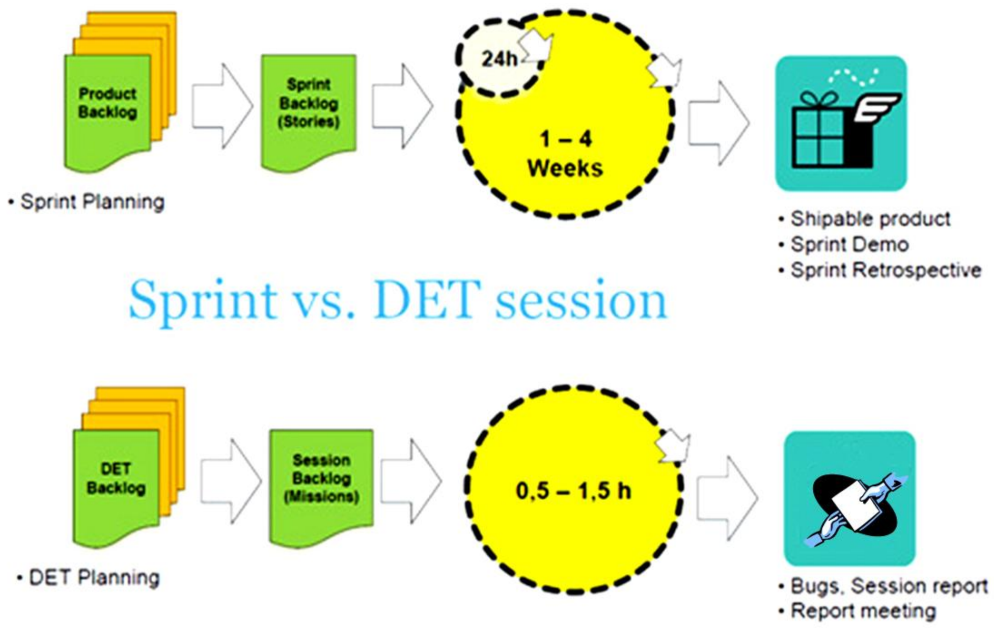


Figure 1: Comparison between Sprint and DET session

As you can see, both Sprint and DET sessions are run in time boxes; for Sprint it is 1 - 4 weeks and for DET session 0,5 - 1,5 hours. In Scrum we have product backlog with all features we need to implement in order to finish the product and deliver it to the market. In every Sprint we take some of those features and put them in the Sprint backlog, the features we are committed to do in the actual Sprint. We have also DET backlog – all features we need to test to have a good test coverage and we take some of them to our session backlog (these are our mission statements). In Sprint we have daily meetings, in DET, since the session is so short, we have a meeting afterwards. We present our findings, what we did and what we found, but also all impediments and what more we need to do.

Main characteristics of DET are:

- **All team members participate in testing activities**
- **Team members are always testing in pairs**
- **Working with different test partners is encouraged**
- **Everybody starts at the same time**
- **A report meeting is held after the test session with all team members**
- **Solution discussion is encouraged**
- **Agreement is achieved on all issues**

All team members participate in testing activities

Since we all share responsibility for software quality we all need to contribute to it. All team members will take part in test sessions and participate in test report meetings including Scrum masters. Even product owner and customer representatives are encouraged to participate in testing activities.

Team members always test in pairs

Testing in pairs, similar to pair programming, is very powerful and effective and very good for knowledge transfer. On the other hand, it mitigates risk when the developers test their own feature alone and we do not need to trace who did what. It is permitted for developers to test their own features with another person who was not involved in implementation of those features. The rule of thumb is that developers who do pair programming will not test together.
Further more, you use your test partner as a sounding board and this, in turn, enhances confidence in your findings; many testers do not raise an issue if they are not really sure it is a bug.

Working with different test partners is encouraged

There are always people who can perform a little better than others. So it is with testing. Some team members are better at testing than others. Changing teams so that you do not always test with the same partner encourages learning from other people and they can learn from you. Everybody has their own tricks and pay attention to different things. So another benefit from changing test partners is by introducing variations in testing. People may test the same feature in various ways and can see completely different things.

Everybody starts at the same time

Test teams start at the same time but they do not need to finish their testing at the same time. However test sessions will be planned to take no more than roughly an hour.

Solution discussion is encouraged

Since developers are doing testing, it is inevitable that they start talking solutions when they discover an issue. This is not prohibited but, on the contrary, it is encouraged. We want to solve all important

issues so this is a good opportunity to discuss issues with other team members. This is done both during test sessions and also the test report meetings after the test sessions. It is however important not to spend too much time on it. Keep it to a maximum of a couple of minutes per issue during test sessions and test report meetings.
It is of immense value when a developer says that they know what the issue depends upon and how to fix it. We can save a lot of investigation time by discussing bugs within the team.

The report meeting is done after the test session with all team members

Everybody who participated in the test session will also participate in the test report meeting but also other team members who did not take part in the test session will listen in and give their comments during the meeting. However, it is not necessary for all team members always to participate in test sessions – it is enough sometimes to have only one test pair. This meeting is very similar to a daily Scrum meeting. Testers (developers who did testing) present their findings, issues and improvement suggestions. They also report if they had any problems and whether they did not test everything which they were supposed to test.

Agreement is achieved on all issues

During the report meeting, the team must agree on all issues. Is it a bug or not? How important is it? In this way we report only issues which we all agree are issues. It could be work in progress, by design, unimportant issues that would never be fixed and so on. Our goal is to give feedback to the actual quality of the product.

Reporting (Test documentation)

Depending on project context you can produce test documentation or not. Customers can demand some kind of proof that you ran testing during your development phase. In such a case, testers can write reports in run time, during the test session. We have our own template, but you can also use James Bach's template for SBTM (Session Based Test Management) Charter.
If you do not need documentation then post-it notes will do.
Every team gets a post-it note with mission statements (features/risks to test). Testers write an issue description on post-it notes; one issue - one post-it note.
At the report meeting when presenting findings, if an issue is accepted by the team, the post-it note is placed on the Scrum board. Otherwise it is thrown away.

Regression Testing

Several people asked me questions about regression testing. What is regression testing in DET? How do we do it?

In traditional scripted testing (both manual and automated) regression testing is most often defined as reusing of previously created test cases. In DET we define it as retesting of previously implemented features.

In DET, we do not have detailed test specifications in order to decrease documentation maintenance cost and benefit creativity. The most important factor of successful testing is domain knowledge and team members certainly have good domain knowledge. Without a detailed test specification they are forced to use their brain much more. Directly after domain knowledge come skills, experience, intuition, motivation and creativity as important factors of successful testing.

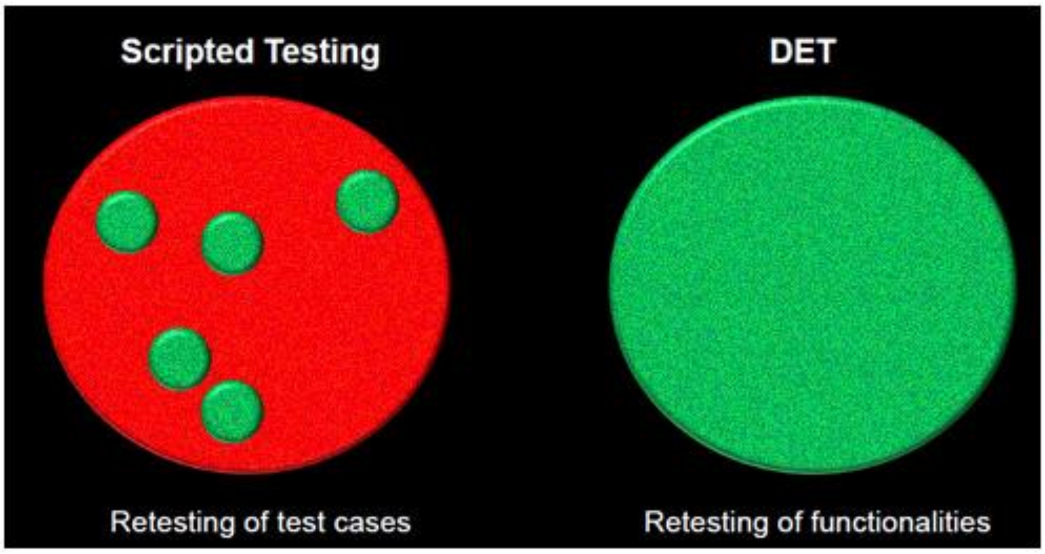


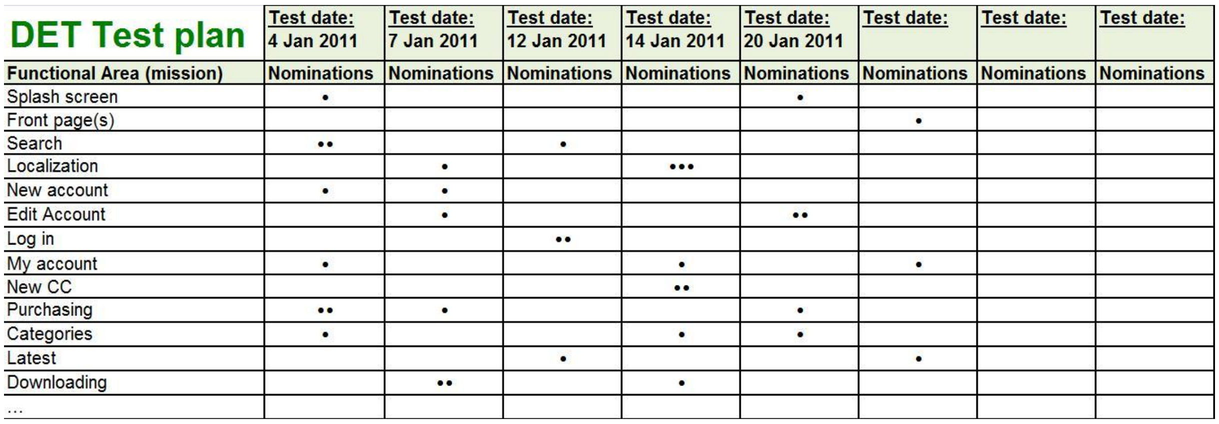
Figure 2: Traditional vs. DET's view on regression testing

The picture above is symbolically showing Scripted testing vs. DET's view of regression testing.
E.g. we have created 5 test cases to test a functional area which we run again and again. Functions which are tested by those test cases are probably well tested (marked with green circles) but it is not difficult to see that the rest of the functional area is not at all tested.
Once, at one of my seminars, I asked one person: "If I give you 5 test cases and tell you to do testing, how long will you do testing?"
He answered: "Until I have tested all 5 test cases".
Then, I asked him pointing at the symbolical functional area on the screen: "And how long would you do testing if I asked you to test this whole functional area?"
He pondered a while then answered: "Until I have no more test ideas and I am satisfied with my testing."

And that is the difference. Of course we do not have 100% test coverage as you might believe looking at the big green circle on the right side. Testing can never be completed. But testers use their creativity when retesting of functional areas/features. Different testers test features in different ways, even the same testers may test the same features in different way on another occasion (they probably learned something new since their last tests) so we introduce a lot of variations and dramatically increase the probability of finding new issues in the same functional area.

It is important to mention that all testers are not equally good at testing but we need to trust people in the team to make their best efforts.

Members of one maintenance team who had a Kanban approach asked me to help them in defining what to test and to plan DET test sessions. I created an excel sheet that looked like the example below.



DET Test plan	Test date: 4 Jan 2011	Test date: 7 Jan 2011	Test date: 12 Jan 2011	Test date: 14 Jan 2011	Test date: 20 Jan 2011	Test date:	Test date:	Test date:
Functional Area (mission)	**Nominations**	**Nominations**	**Nominations**	**Nominations**	**Nominations**	**Nominations**	**Nominations**	**Nominations**
Splash screen	•				•			
Front page(s)						•		
Search	••		•					
Localization		•		•••				
New account	•	•						
Edit Account		•			••			
Log in			••					
My account	•			•		•		
New CC				••				
Purchasing	••	•			•			
Categories	•			•	•			
Latest			•			•		
Downloading		••		•				
…								

Figure 3: An example on DET test plan. Similar test plan was printed and hung on the wall. Nominations and dates were inserted by the team.

The first column is a list of all features of the product (functional areas). When a developer fixes an issue a dot is drawn in the feature raw in the coming test session. Then they also analyze what other features could be affected by the fix and draw dots in the corresponding places. Developers are self-driven and choose what will be tested next time they run DET.
The more dots a feature has the more thorough testing of that feature should be. If some feature was not tested for a long time it can also be chosen for a coming test session.

Benefits of DET

Some of the main benefits of DET are:

- **There is no need for exhaustive test documentation.**
  Only as much documentation as necessary, so team members have more time for coding and testing.
- **Finding bugs and giving feedback to developers early in development**
  I do not believe I need to explain that finding bugs early in development can save a lot of money.
- **Developers do testing:**
  - They improve their testing skills
  - They get better at coding (learning from their mistakes – known bugs)
  - A smaller team is needed
    It does not mean we have less testers, on the contrary everybody is a tester so actually we have more testers.
  - There is no arguing about bugs. Team members reach agreement on them.
  - Discussing issues at session meetings very often will save time that otherwise would be spent on issue investigation.
- **Testers test in pairs**
  - More, and more interesting, bugs are found (at least we feel so)
    ** First time a dedicated external group of professional testers tested the software my team had delivered, they found 7 issues. Those 7 issues were among 35 open issues which the developers' team already had found. I am not pointing that developers are better testers than testers but they are certainly closer to the software and have excellent domain knowledge which is essential for the effective testing. Professional, dedicated testers who work in the team together with developers are great at findings bugs because they have both good domain knowledge and testing skills.*
  - It is good for knowledge transfer

- Less risk developers test their own feature alone (more objective testing)
- It is much more fun; developers really like it

Summing it up

Developers do testing from the beginning of the project; get better at testing and at the same time better at development. They are more enthusiastic about testing and actually **LIKE** testing. They find more bugs and the quality of the product is frequently discussed in the team so that there is enough information to plan what to do next.

Davor Crnomat, @Jayway