GRANULATED TESTING. TNT FOR THE PEAK OF THE HILL

Sergei Zhuk

Quality Department of Shopify, Berlin, Germany

ABSTRACT

This contribution gives a review of granulated a testing approach for JSE 2019 committee.

KEYWORDS

Issues, defects, bugs, test management, test planning, development planning, waterfall, sashimi testing.

1. PROBLEM STATEMENT

This story is based on my last 5 projects with strict deadlines and high requirements to product quality. Initially we used the following workflow: build stage is over and our team starts testing, in other words we usually start feature testing when majority of development tasks are completed. This type of workflow caused particular limitations. I'd like to describe those limitations and the solution we manage to find together with project leads. The main issue turned out to be unforeseen peak of defects number in the middle of system test stage.

So for this project in Figure1 you can see what I'm talking about: build stage is over and we start testing then all of a sudden, in the middle or what is even worse - in the end of system test stage a huge amount of issues appears. And most of these issues are show stoppers.

All the figures you see in this presentation were built on data from one of my recent projects. It took us 6 weeks from the start of the build to the end of the internal system test. The project's team included 1 Dev Lead, 7 Developers, 1 QA lead, 4 QA Engineers.

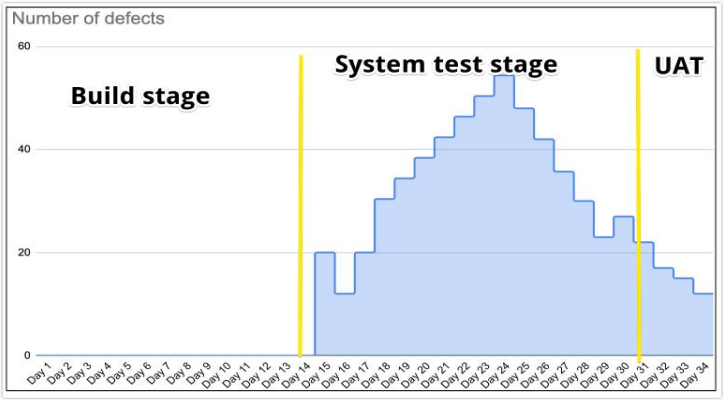

Figure 1. Blockers and critical defects distribution for common testing approach

Natarajan Meghanathan et al. (Eds) : NLP, ARIA, JSE, DMS, ITCS - 2019
pp. 81-87, 2019. © CS & IT-CSCP 2019 DOI: 10.5121/csit.2019.91210

Does this look familiar? This is too much familiar to me and I must confess that release management starts looking like some obstacle race. Do you want to know why? Let me show you:

- Testing team becomes blocked by this peak of issues at the hottest point of testing - right after the middle of system test stage. And QA engineers have to waste their time waiting for the fixes of blockers. In our case, when the number of defects exceeded the number of 40 then the whole testing process gets blocked.
- Senior developers on whose behalf an issue is created will not be assigned to fix it later. The issue will be fixed by some middle or junior developer. And we can't resolve this on the stage of release planning because we don't know how many issues we'll find and how many resources the bug fix will require. I often hear the management saying: “After all, now we have a system test stage and bug fix can be supported by middle and junior developers”.
- Finding the root cause (RC) is a crucial point and it takes pretty much time because of the following:

a) The system gets very complicated by the end of the build. The final model has a lot of attributes, dependencies, integrations etc
b) People who implemented features were moved to other activities
c) The price of fixes after the release is higher than the price of fixes during the build phase. It is obvious.

- A real nightmare of the test team is a situation when, on the stage of system test, it turns out that a redesign is required.
- Due to the heap of blocking defects a part of the system remains untested.
- We are not able to estimate the quality of our product at build stage. E.g. in our example it’s not so easy to track the quality of work being done during the first 3 weeks (40% of the release life time).

As a result we have what we have:
- Vague delivery dates at each phase
- Testing is blocked by the peak of defects.

At this point we start burning our people and the budget. The product is delivered to the customer with a certain amount of known issues and the uncertain amount of unknown issues. Do we want to deliver a product whose quality is under a big question? No, we don't. We want our customers to be happy with the quality of the product we produce. That's why we started thinking: "What kind of TNT can we use to remove the peak so we wouldn't even have to start climbing to the pinnacle?" We started looking for ways to resolve this issue and it looks like we've managed to find a solution

2. RELATED WORK

We investigated different ways how to eliminate points above. Our choice fell on “sashimi” testing. It was initially described in Steve McConnell book “Rapid Development: Taming Wild Software Schedules.” We started to experiment with “sashimi" testing.

SOLUTION DESCRIPTION

We've found the TNT we were looking for. We called it Granulated Testing. Now, let me introduce my idea of granulated testing in detail. What can be considered a granule in software development? Commits are the granules. Commits.

And we believe that each granule should be tested. Each commit should be verified in the same revision it was committed. The bad news is that this approach requires reviewing and changing the current development planning process. The development process should include verification and validation of every dev task commit. Of course we should be careful implementing this approach.

One of the main paradigms of saving a budget and successful product delivery is to start testing as early as possible. It should be started prior to the e2e completion, prior the full functionality is implemented. It should be started even when such epic's features like forms and pages are not ready yet. We would strongly recommend to start testing as soon as first commit is done.

Yes, it is not easy. Yes, it requires extra planning and extra effort from Dev Lead and QA Lead. BUT it will help us to get rid of that peak due to more even distribution of issues. The granulated testing method works indeed. I will now describe the conditions when it’s applicable and what kind of preparation work is required:

2.1. SHOULD SPEND MORE TIME PLANNING DEVELOPMENT AND QA:

- Consider task dependencies from a development point of view. Note: the tasks with dependant sub-tasks should start the earliest. For example, start the configuration of the object model as soon as possible because it is a prerequisite for build integration. Divide complex processes into simple steps and deliver plain independent parts prior to the feature testing.
- QA lead should review the release plan paying attention to testability of developers' tasks. Note: at this point QA lead should consider all possible ways to test the commit - whether to test it using DB clients or fire request directly to web service’s endpoint and so forth.
- Take into account the scope of the delivery and think when QA engineers should be involved for granulated testing. E.g. in my project I started testing along with 1 additional QA from the 1st day of the build stage.

2.2. WE SHOULD BE READY TO START TESTING EARLIER.

Yes, it’s obvious but nonetheless: we should prepare testing environment in advance as you understand prior to the first day of the build stage. It’s good to have a stable continuous integration system for deployment because we are going to receive new commits for testing several times a day. Note: QA team doesn’t need an ideal build, a build that can be installed without blockers is enough.

2.3. We Should Understand How A Particular Commit can be Tested.

It will be definitely different from feature testing.
Example:
if
dev task for commit = "Add street attribute for Address Form".
then,
Test = validate required tables in DB.

2.4. Requirements.

Major feature requirements should stay unchanged during the build stage. If it is not possible to have them stable, then the solution I'm talking about will not be applicable.

2.5. Kick-Off Meeting.

We should have a kick-off meeting with project management and developer lead prior to the release planning to discuss if the suggested solution can be used or not. Also consider all the pros and cons and decide if it is worth to apply the suggested method.

So let's discuss the pros and cons

3. Pros and Cons of the Proposal

3.1. Let's Start with the Cons.

First of all, early start of testing leads to early start of budget eating because test team gets involved earlier. But we will use the budget smarter. We'll eat our elephant by small pieces. And by the end of the project we all will be full and happy. This of course should be communicated to the management in the right way during the release planning stage.

Another disadvantage of the approach we suggest is the necessity to introduce particular changes into the development process, it should provide the ability of every commit verification - the commit should be testable. The verification should be reasonable, without fanaticism. After all, we want to make our life easier, we don't want to burden ourselves with excessive bureaucracy.

That's why we would expect experienced Dev leads and QA lead who is qualified enough to do the planning taking into account our suggestion to start testing as early as we can. Someone who could see the main concept of the product at its very early stage. Is it too much to ask for? Not at all! We have to find the solution of resolving issues without delivery delays.

3.2. Now Let's Consider the Pros

3.2.1. The quality of the build can be tracked from the very first day of the build. During release planning we can file all developer tasks(=commits) into a tracking system e.g. Jira. To visualize the development and testing progress for project management, dev and qa leads. If we get delays we'll be able to reveal them earlier and therefore, we have more time to build a mitigation plan.

3.2.2. The test team finds defects earlier, the development fixes them earlier as a result it the cost gets reduced.

3.2.3. Even distribution of defects helps to plan the operation of QA command minimizing team's outage.

3.2.4. The test team sees the whole picture of the system and understands its compounds a.k.a. granules. QA team understands the purpose of all objects of the product, as well as dependencies, integrations etc. This helps to write more correct test scenarios and cover all the functionality. It often happens that an experienced QA engineer knows what exactly is broken and how it can be fixed.

3.2.5. The main advantage of this method is that the dev team gets a better disposition for a fix.

Example: An end to end scenario does not work, then developers look into the defect's description and see that the object model has been validated in a particular dev task and all they need to do is just to verify integration part. From issue's lifecycle track it is clear in what build revision the issue was not present and developers can understand either they need to consider narrower areas of code responsible for integration or some particular commits.

When developers see the whole defect's story they spend less time on root cause investigation. And it is even easier to attract any developer to fix this bug.

And so forth... Using this approach we've managed to decrease the amount of time spent on root cause analysis three times.

4. SUMMARY AND RESULTS

The method described above works the best for projects with waterfall model. Our granulated testing solution is based on the “overlapping stages” idea a.k.a. “sashimi” method. We've been testing it for a while and came with our own adjustment based on the results of regular "Lessons learned" reviews.

Now, take a look at Figure 2. This is an illustration of the effect our solution makes. Look at this red graph and compare it with the blue one.

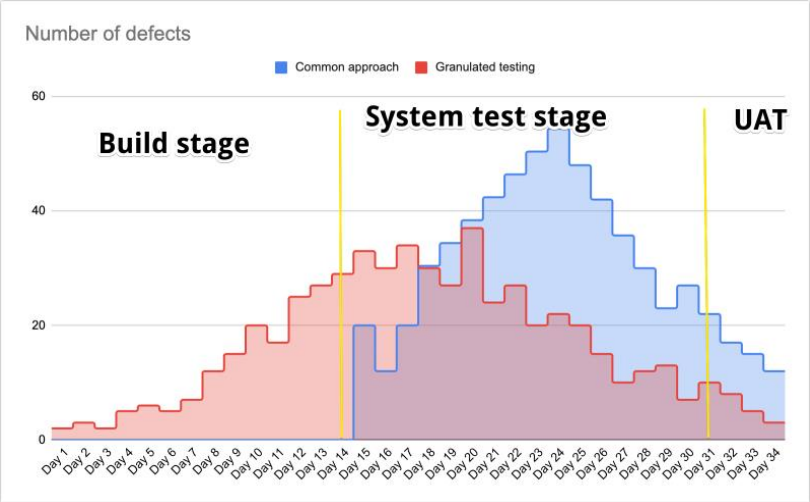

Figure 2. The comparative defect distribution during release life time

RESULTS

4.1. THE PEAK OF DEFECTS

These methods into developing a solution that helps people log their food and calories and ultimately helping them form a habit of recording their meals.

4.2. TEST TEAM OUTAGE

As a result, we decreased the amount of time when QA team is blocked.

4.3. RELEASE TRACKING

Management, dev lead and QA lead are able to track release quality from the first day of the build stage.

4.4. NUMBER OF CRITICAL AND BLOCKERS

We've also managed to reduce the number of critical issues in the end of system stage phase twice.

4.5. BUDGET

The cost for development and testing has reduced for 5-10%. Important note: we got this benefit starting from the second release. Please take a look at Figure 3

Common approach	100%
1st release with granulated testing	106%
2nd release with granulated testing	98%
3rd release with granulated testing	95%
4th release with granulated testing	93%

Figure 3. The comparative cost of development and testing (from release to release)

As you can see, the cost of the first release increased. This was due to the extra effort we had to make in order to introduce the method:

- It took us some time to learn how to perform the planning, what artifacts and environment should be prepared prior to the beginning of granulated testing
- We also had to modify communication workflow between the teams of developers, QAs and managers.

The granulated testing approach was successfully applied to 5 OSS (operational support system) projects. Those projects took 2-3 months. The whole team consisted of 20-30 people (approximately 5 QA engineers). I believe that the approach I told you about can be applied to waterfall projects meeting the requirements described above.

ACKNOWLEDGEMENTS

The author would like to thank Tatyana Pirogova.

REFERENCES

[1] McConnell, Steve (1996). Rapid Development: Taming Wild Software Schedules. Microsoft Press.

[2] Kaner, Cem; James Bach, Bret Pettichord (2001). Lessons Learned in Software Testing: A ContexDriven Approach. John Wiley & Sons.

[3] A Waterfall Systems Development Methodology … Seriously? by David Dischave. 2012.

AUTHOR

Sergei Zhuk Experienced Quality Assurance Lead with a demonstrated history of working in the telecommunication, OSS/BSS industries, and ecommerce. Skilled in Waterfall, Agile Environment, Scrum, SQL, Requirements Analysis, Agile Methodologies, Audit of testing processes, Testing improvements, and Test Automation(Selenium, Testcafe, SOAP UI) for different areas: Smoke testing, UI testing, API Testing including configuration and integration with CI(e.g. Jenkins). Strong quality assurance professional with a Programmer background(Java, Nodejs, Shell). ISTQB Certified Test Manager. Good experience as UAT manager around the World.