МАТЕМАТИКА
MATHEMATICS

UDC 004.051

L. Giorgadze[1], D. Alibiev[1], G.De Chirico[2], A.Sh. Kazhikenova[1]

[1] *Ye.A. Buketov Karaganda State University, Kazakhstan;*
[2] *Alpegagroup, Belgium*
(E-mail: silverluka@mail.ru)

Minimization of product defects by the means of implementing correct test pyramid in the software development process

Article attempts to present one of the solutions towards the problem of flawed and delayed product development. The history behind current most popular product development methodology (waterfall methodology) is traced, as well the problems that this methodology presents such as calendar risks or high employee turnover rate. The methodology that was created to solve those issues is gaining a lot of popularity now, because it offers much more agile way of delivering a software product by splitting the whole development process into manageable iterations. Customer can change the requirements and adapt the product to the market changes from iteration to iteration. However, this new methodology (agile methodology) presents some serious problems as well. This article concentrates tackles the main issue – insurance of the product quality with each iteration. The solution that is presented by the article revolves around creating and maintaining a correct test pyramid, with the specific description of all the layers of the pyramid. A small study was conducted on a sample project, where it was calculated, that test pyramid allowed to decrease the amount of defects by 25 %.

Keywords: software development, agile methodology, test pyramid, waterfall methodology, unit tests, integration tests, system tests, acceptance tests.

Every customer is interested in a development process that would require least investment. This interest has initially caused an emergence of the waterfall model [1], which sets a complete set of requirements and deadlines, in a span of several consecutive stages:

– Analysis and determination of the product requirements, which results in the creation of the product specification (also known as planning phase);
– Design of the product;
– Development of the product;
– Product testing;
– Fixing of defects that were found during the testing phase;
– Installation;
– Support.

Having dedicated phases yields manybenefits theoretically – every phase has specific time frame, therefore it should be easy to manage and calculate necessary resources. However, this methodology introduces quite a few risks [2]. It is generally agreed that there are five main risks:

Calendar risks, or planning mistakes

It is very hard to give precise time and resource estimations for the process of software development, because many factors have to be accounted for: time spent on learning the technology stack, sick leaves, staff turnover rate, the stability of all machines, etc. Besides this, sales managers can try to lure the customer by initially providing an overly optimistic time and resource estimations, or they could provide an erroneous estimations due to lack of knowledge in a specific field.

Changes in the product requirements after the planning phase

Often times, the customer changes his opinion on how the product should look like, which results in changes to the requirements. Usually, the bigger the product, the more changes will be done to the requirements after the planning phase was concluded. This leads to two options:

a) Include some time and resources for potential requirements changes in the initial estimations. However, how much time should be allocated for changes that are not known at the planning phase?

b) Prohibit making any changes to the requirements. However, what if the customer desperately needs some major modifications due to, for example, recent market changes? Inability to adapt to new needs would leave the customer dissatisfied.

Employee turnover

There is no guarantee that the initial team composition will persist throughout the whole development phase – leaves are unavoidable. Leaves of experienced employees, who are familiar with the goals, requirements, tasks, product architecture, technologies, who have established efficient communication chains with other team members, will severely affect the development process. Furthermore, such cases would make it necessary to spend time to find a suitable replacement and introduce that person the product. Figure 1 shows the economic effects of high employee turnover rate.

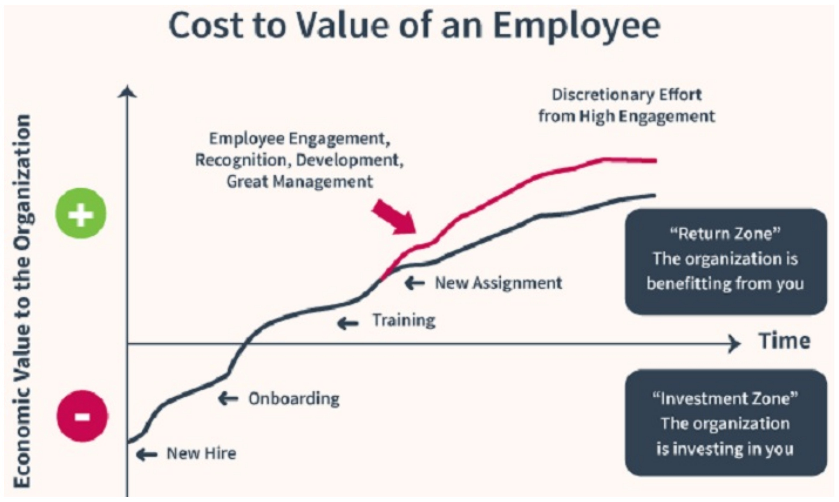

Figure 1. Representation of the economic effects of employee substitution

Mistakes/inconsistencies/flaws in the specification

The process of writing a specification is quite long and grueling; therefore, it is extremely difficult to avoid mistakes. Some flaws of the specification might be identified deep into the development phase, where it is quite costly to fix them.

Fluctuating productivity

Team productivity and productivity of each individual team member is not a linear, but rather a dynamic value as it is affected by many factors. These factors could be work related (team atmosphere, relationship between the customer and the development team), or personal (some special circumstances in the lives of the team members, their motivation, etc). According to Parkinson law, the development team reaches its top velocity towards the end of the development phase, while working at about half of its potential during the majority of the development phase [3]. Therefore, if the estimations were done with the regards to the top velocity of the team, they will be erroneous. Tom DeMarco and Timothy Lister in 'Waltzing with Bears: Managing Risk on Software Projects' provide peculiar statistics: if the initial estimation for a product was 26 months, probability of meeting that estimation is 4 %. With 75 % probability work will be done in 38 months, and in 15 % of cases the work will not be completed whatsoever [4].

In order to avoid these risks many projects have started to use the agile methodology, with which the development process consists out of manifold short cycles that are called iterations. Each iteration is two to four weeks long and is a miniature full development process, consisting of all phases, described in the beginning of the article. Using such an approach provides the following benefits:

1. More precise estimation of the development cost due to planning requirements only for the next iteration, not for the whole product.

Agile methodology aims to eliminate the cumbersome and archaic product specifications. Instead, the customer provides a general idea for the desired product and a special human resource is allocated – the product owner. Product owner is responsible for providing specific requirements for each iteration. Product owner is a part of the development; he acts as a medium between the customer and the developers. As a result, the development cycle of an iteration looks the following way:

a) Phase 1 – requirements analysis. During this phase, the needs of the product are analyzed and requirements are formed. Requirements are stored in a form of an issue in special issue tracking systems, where they are available to every interested person. Each issue should have an informative description (mini specification) as well as acceptance criteria – criteria that will allow to unambiguously discern whether or not the issue was implemented in the project. These issues should be prepared ahead of the iteration by the product owner;

b) Phase 2 – iteration scope estimation. The iteration is kicked off with a meeting, during which the created issues are estimated. The whole development team participates in this meeting, and everybody can make suggestions and ask questions to clarify the requirements. There different ways to estimate the issues, with man-hour and story point estimations being the most popular ones. After all issues have been estimated, product owner chooses the scope for the iteration based on the priorities and the estimations. The fact that the whole team participates in this meeting provides for a more accurate and efficient estimation as different members of the development team (developers and QA specialists) are able to voice all types of concerns and risks for all issues;

c) Phase 3 – development and testing. During the iteration, development and testing of the issues are done concurrently. Test scenarios are being written and agreed upon with the product owner during the development of the issue, and when the issue is completed, these scenarios are used by QA specialists to verify that the issue complies with the acceptance criteria;

d) Phase 4 – demonstration session. The iteration is closed with a meeting, during which the completed issues are demonstrated to the product owner. Product owner can point out things that were missed or suggest improvements immediately. This immediate feedback provides for a product that fully meet the customer needs [5].

2. Keeping the product up to date with the customer needs due to frequent planning phases.

As a result of introducing a product owner role, the customer has its own representative in the development team at all times. This allows better controlling the resources at work and even changing the requirements during the development phase, when the cost is the lowest. Furthermore, product owner has the complete picture of the whole development process, the issues that the development team has, various blockers, etc.

3. Agile methodology does not necessarily solve the problem with employee turnover, but it does present some engineering practices that help reduce the negative affects if such cases do take place.

4. Having the specification/requirements for the product in an issue tracking system.

Giving the development team free access to the whole storage of the existing issues, helps identify issues /inconsistencies/flaws in the specification much earlier. Having the specification split into smaller pieces (issues), provides for an easier way to get familiar with the required part of the product, instead of having to go through the whole specification.

5. Immediate feedback and frequent results.

Having the development process split into many small iterations allows the customer to have a working version of the product at the end of each iteration with the features that were developed during that iterations. Customer can use this product, analyze how it fits, possibly provide changes to better suit his needs, etc, but he does not have to wait for a long period of time to see the result of the work.

Even though the agile methodology solves many problems presented by the waterfall approach, it has its own risks [6]. The main risk of the agile methodology is assuring the quality of the product – the more functionality is added with each iteration, the bigger the product gets, the harder it gets to not break existing features while developing new ones. The process of defects creation in already existing functionality is called regression. Regression forces the quality assurance team to execute regression testing during each iteration, which means that they have to make sure that all key existing features were not affected by the features added during the iteration. Considering the fact that the amount of features grows with each iteration, the amount of regression testing required grows progressively and becomes quite an issue for the quality assurance team. To fit both new feature testing and regression testing into one iteration, the QA team has to descope some issues, or reduce the intensity of testing.

Fortunately, there are various ways to reduce the amount of manual regression testing done in each iteration and in turn reduce the amount of resources required to be allocated for regression. There are practices for both the development and the quality assurance teams. One of the most revered practice is writing and support of tests by the software development team [7].

Each product flaw or defect is a result of a flaw in the product code (unexpected situation or a regular mistake by the developer). Code defects are an avoidable part of the development – it is impossible to fully eliminate them. However, it is possible to minimize such defects and therefore reduce the amount of bugs. To ensure that the code does not contain mistakes, there is a practice of writing test code that tests the code of the product. Test code imitates certain scenarios and checks the behavior of the product code in these scenarios. Each test has the following steps:

– Test data preparation – specific data for the tested scenario is prepared;
– The tested code is executed;
– Results of the code execution are verified;
– Prepared scenario data is deleted.

There are various types of the code tests. Figure 2 depicts the test pyramid [8] which identifies four levels of code testing and puts them into a structured hierarchy. Each level has a corresponding type of tests that are targeted at different aspects of the code development process.

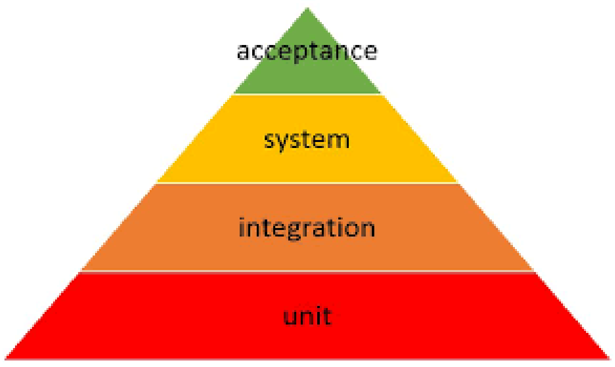

Figure 2. Test pyramide

1) Unit level and unit tests. Product code is atomized to single independent and isolated units. Units tests focus these units separately, testing their behavior in solitude. All interactions between units are mocked – simulated with the desired result for each particular interaction. This secures the fact, that all units' logic is not affected by the outside units. These tests are lightweight because they test only small pieces of the application, easy to write and support as they are limited to testing only small parts of logic. Due to these factors, it is possible to cover the majority of the product's code base with such tests, and run these tests during each build [9].

2) Integration level and integration tests. While unit tests guarantee that units work in isolation, integration tests verify how these units interact between one another or with some outside components. For example, an integration test could verify how data is read and written to a database – in this particular case we test that product code, responsible for connection and communication with the database performs its functions well. We cannot conclude whether or not the code works without testing it with a real database component. Obviously, interaction with an outside database component is quite resource-intensive compared to the unit tests, therefore the number of these tests is considerably lower than that of the unit tests.

3) System level and system tests. This level is responsible for testing how all the components work together in an environment similar to a production environment (without any isolation or simulation of outside components). Systems tests verify that all the units and outside components work well together validate the database state during the logic execution, check the correctness of the interactions between the components. Such tests are quite resource-intensive and take longer amount of time to fully

run. Similarly, the amount of such tests is lower than that of integration or unit tests, because those tests cover considerable amount of logic, while system tests make sure that the product works well as a whole mechanism.

4) Acceptance level and ui tests. While previous three levels focus on how the code of the application corresponds to the requirements, acceptance level is responsible for verifying the interaction between the functionality core of the application and the user interface. UI tests imitate user actions using the scenarios that reflect what regular users are most likely to do in the application. These tests are the most resource-intensive tests as they use the production environment (or an environment that replicates production environment). The amount of these tests depends on how many most used user-cases there, but usually, there are not too many of such tests. Such tests are not usually run with each build [10].

Besides providing a safety net against regression issues, unit tests help reduce the cost of identifying and fixing bugs. Figure 3 illustrates the cost to bug identification time ratio – it can be seen that correcting issues that are caught by unit tests is much cheaper, then correcting them after QA feedback.

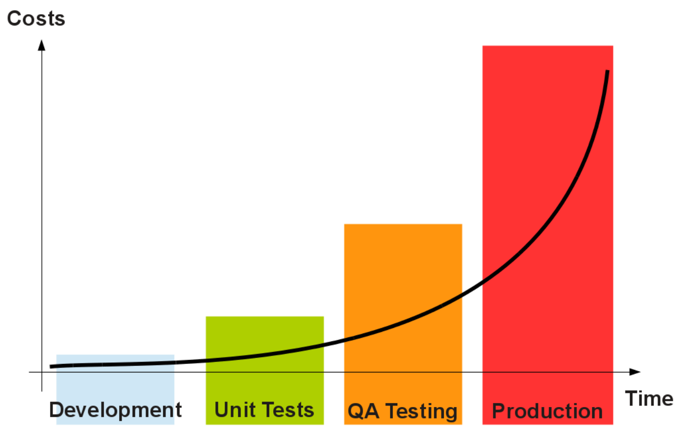

Figure 3. Relation of time to defect fix cost

Addition and maintenance of tests to the software development process allows to significantly reduce the amount of defects in the product, which provides for improvement of the user experience and therefore maximizing the profit of the application. A sample moderately small project was monitored over a period of 6 months before unit tests were added to the development flow, and after. Figure 4 displays the amount of defects per month before the test pyramid was introduced and implemented on the project.

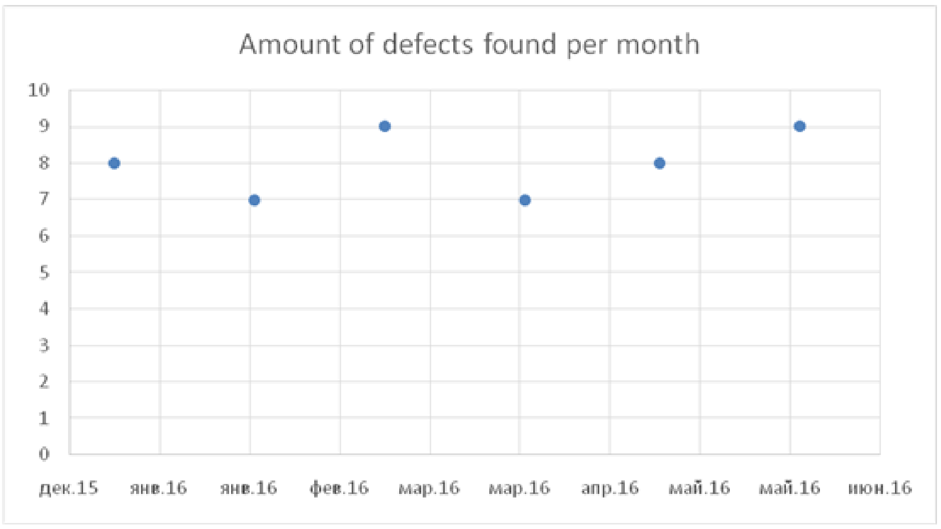

Figure 4. Amount of defects per month before test pyramid was introduced

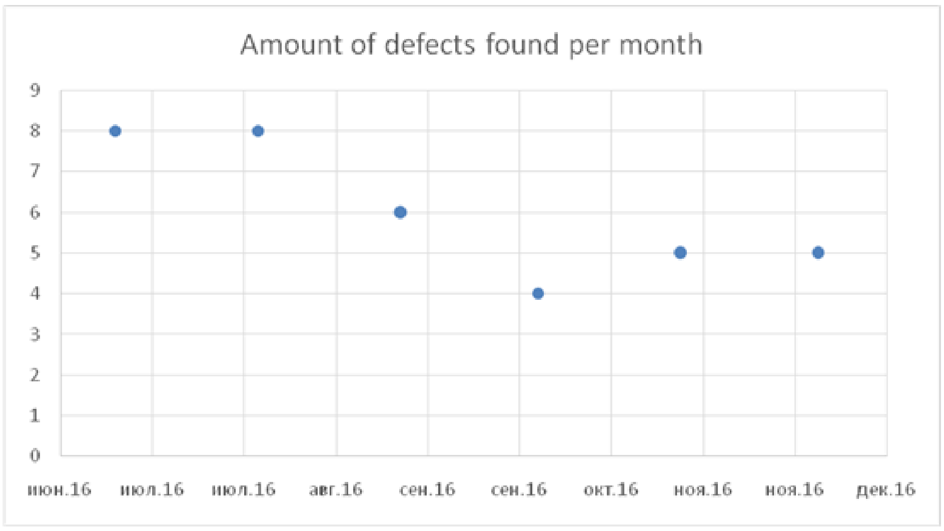

Figure 5. Amount of defects per month after test pyramid was introduced

As we can see from Figure 5, the average amount of defects found per month gradually decreased from 8 to 6, which is a 25 % decrease.

References

1 Petersen, K., Wohlin, C. & Baca, D. (2009). The Waterfall Model in Large-Scale Development. Product-Focused Software Process Improvement. PROFES 2009. Lecture Notes in Business Information Processing, Vol. 32, 386–400. Springer: Berlin, Heidelberg.

2 *atlaz.io.* Retrieved from https://atlaz.io/blog/the-risks-of-waterfall-methodology/.

3 *projectmanagementlearning.com.* Retrieved from http://www.projectmanagementlearning.com/what-is-parkinsons-law-in-project-management.html.

4 DeMarco, T. & Lister, T. (2003). Waltzing With Bears: Managing Risk on Software Projects. USA: Dorset House.

5 Larman, C. (2003). Agile and Iterative Development: A Manager's Guide. USA: Addison-Wesley Professional.

6 *scrumalliance.org.* Retrieved from https://www.scrumalliance.org/community/articles/2014/july/risks-in-agile-projects

7 *martinfowler.com.* Retrieved from https://martinfowler.com/bliki/TestPyramid.html

8 Cohn, M. (2009). Succeeding with Agile: Software Development Using Scrum. USA: Addison-Wesley Professional.

9 Koskela, L. (2013). Effective Unit Testing: A Guide for Java Developers. USA: Manning Publications.

10 Cimperman, R. (2006). UAT Defined: A Guide to Practical User Acceptance Testing. USA: Addison-Wesley Professional.

Л.Гиоргадзе, Д.Алибиев, Дж.Де Кирико, А.Ш. Кажикенова

Бағдарламалық қамтаманы құру барысында нақты тестілік жүйені қолдау жолымен өнімдік ақауларды болдырмау

Мақалада жасалған бағдарламалардың жетіспейтін жерлерін және уақытша тоқтап қалу мәселелерін шешу жолдары қарастырылған. Осы айтылған мәселелерді шешудің бағдарламалық қамтаманы құрудың көрнекі әдістемесінің, яғни сарқырамалы әдістің, пайда болу тарихы мен жолдары көрсетілген. Сарқырамалы модельді мәселені шешу үшін құрылған әдістеме, бағдарламалық қамтаманы құру процесін басқаратын бірнеше итерацияға бөлу арқылы бағдарламалық қамтаманы құрудағы икемді тәсіл болып табылатындықтан, қарқынды дамуда. Тапсырыс беруші нарықтағы талаптарға сәйкес ақырғы өнімді әрбір итерация барысында өзгертуге және бейімдеуге мүмкіндік алады. Алайда бұл жаңа бағдарлама құрудағы икемді әдістеме бірқатар мәселелерді шешуді талап етеді. Бұл мақалада осы мәселелердің ішіндегі ең негізгісі болып табылатын, бағдарлама құру кезіндегі әрбір итерациядағы өнімнің сапасын қамтамасыз ету мәселесін қарастырамыз. Авторлар ұсынған мәселені шешу жолы осы жүйенің әрбір қабаттарын көрсету және сипаттау арқылы дұрыс құрылған тестілік жүйенің (пирамиданы) құру және оны қолдауға негізделген. Жоба мысалында жүргізілген зерттеу нәтижесінде тестілік пирамида ақауларды 25 %-ға азайтуға болатынын көрсетті.

Кілт сөздер: бағдарламалық жасақтаманы әзірлеу, икемді әдіснама, тестілеу пирамидасы, сарқырамалы әдістеме, юнит тестілері, интеграциялық тестілер, жүйелік тестілер, қабылдау тестілері.

Л.Гиоргадзе, Д.Алибиев, Дж.Де Кирико, А.Ш. Кажикенова

Минимизация дефектов программных продуктов путем поддержания корректной тестовой пирамиды в ходе разработки программного обеспечения

В статье представлен один из возможных путей решения проблем, возникающих при разработке программных продуктов, имеющих недочёты, а также связанных с временными задержками. Рассмотрены история возникновения наиболее популярной методологии разработки программного обеспечения – водопадной методологии, и те проблемы, которые данная методология представляет. Методология, которая была создана для решения проблем водопадной модели, набирает всё большую популярность ввиду того, что она позволяет более гибкий путь создания программного обеспечения за счёт разделения всего процесса разработки программного обеспечения на небольшие управляемые итерации. Заказчик может менять требования и адаптировать итоговый продукт к требованиям рынка от итерации к итерации. Однако эта новая методология – гибкая методология разработки – ставит ряд новых проблем. Данная статья рассматривает самую главную проблему – обеспечение качества продукта с каждой итерацией разработки. Решение, предложенное в статье, основывается на создании и поддержании правильной тестовой пирамиды, с указанием и описанием всех слоёв данной пирамиды. Было проведено исследование на примере проекта, которое показало, что тестовая пирамида позволила уменьшить количество дефектов на 25 %.

Ключевые слова: разработка программного обеспечения, гибкая методология, тестовая пирамида, водопадная методология, юнит тесты, интеграционные тесты, системные тесты, приёмочные тесты.