CHECKLIST

How do you know if your project is in danger?

- *Instruction for working with the checklist*
- *Detailed descriptions of different types of dangers and risks that affect the project*
- *Tips and recommendations on how to handle these risks*

Dzmitry Harachka

IT consultant / Founder and owner of a software development service company

Instruction for working with the checklist

Psychological signs

1. You often hear from your developers: “It's hard, it's impossible. It will take years" [max: 3]
2. To the question: "Why is it difficult to implement?" — developers always have excuses [max: 3]
3. Your developers are passive and not interested in the project [max: 3]
4. You know that one of your engineers implemented a cool improvement, brought real value to the project, but it was never applied [max: 3]

Technological features

Technical debts

5. There is no established practice of regular processing of technical debts [max: 5]

Integration and Delivery

6. The build of your project cannot be built using a single console command [max: 10]
7. Founders (you) are not willing to invest in automating the product delivery process [max: 3]
8. There is no full-fledged CI/CD on the project or not all stages of this process are automated [max: 5]
9. Delivery times are missed on a regular basis [max: 3]

Writing and testing code by developers

10. There is no conscious, well-formulated and written strategy for working with various branches of the application's source code. Plus, there is no understanding of how to organize the source code of different versions of the product. The process of creating new features is uncontrollable. All this suggests that the project does not have a well-developed Git flow [max: 3]
11. No unit tests, autotests for:
    a) Server part of the system [max: 3]
    b) UI parts of the system [max: 4]
12. There are no unit test code coverage metrics that are understandable and accessible to the entire team [max: 2]
13. Your developers don't test their changes on a separate environment before they merge their code into a shared branch [max: 3]
14. Your developers do not run autotests before merging with the main branch (locally or separately on a dedicated server) [max: 4]

Testing

15. The core of your product is not covered by unit tests, while the power of the set of input values is very large [max: 7]
16. There are no documented test cases for manual testing. [max: 3]
17. Special test plan management tools are not used (TestLink, TestRail) [max: 3]
18. No connection between test case sets for manual and automated testing [max: 1]
19. Unforeseen bugs in production that pop up systemically. There is a feeling that you cannot stabilize any part of the product in any way [max: 2]

Immersion of new team members in the project

20. There is no clear textual description of the architecture, as well as the key features and processes of your project [max: 3]
21. Your system has a large codebase, while there is no playground (reference project) where novice engineers can dive into the project [max: 2]

Problems with management and processes

- *22. Your manager is not interested in and understands the technical side of the project. CI / CD and control of Unit tests mean nothing to him [max: 5]*
- *23. No well-tuned project management system [max: 2]*
- *24. Estimation and task decomposition system not implemented [max: 2]*
- *25. No description of project benefits for the team [max: 1]*
- *26. Your team works remotely, but does not turn on cameras at stand-ups [max: 1]*
- *27. You don't attend demos and stand-ups in person [max: 2]*

---

*Hi! My name is Dzmitry Harachka. I am an IT consultant on the implementation of Agile/ Scrum and CI/CD in product development and delivery processes. I have more than 17 years of experience in Java development and Scrum management of international projects. I am also the founder and owner of JazzTeam, an **IT service company**.*

Being a founder myself, I know perfectly well all the issues and challenges you might face when creating IT solutions. Perhaps, your project has made good of it or has just started out, is in the beginning of its journey. As a company owner or a manager, you make important decisions to develop the company, build the business strategy and move the product forward. But as time passes, you may realize or understand intuitively that something is going wrong on the project. At the same time it can be very difficult to determine the real and underlying reasons on your own.

*To understand **if your project is in danger**, please follow **the checklist** that I developed based on the experience of numerous projects in which I participated as a developer, consultant, and manager.*

Instruction for working with the checklist

1. **How to assess the situation on the project?** Below you will find detailed descriptions of different types of dangers and risks that affect the project. The information is structured along three axes: psychological signs, technological signs, problems with management and processes. Each item contains a detailed description of the risk; information related to the causes and possible consequences of problems; a block with brief tips and recommendations on how to handle this risk. Also, each item corresponds to a certain point value (the highest point value is indicated in square brackets after each question).

2. **How to calculate the total score?**
   To assess the degree of danger, you need to rate each risk on your project and sum up the all points. In this case, the point values can be interpreted as follows:
   - **The highest point value** - this exactly corresponds to the situation on your project.
   - **0 points** - this is absence on your project.
   - **Points between 0 and the highest point** value means that such issues exist, but partially. Or they have not yet reached the critical point.

3. **What does the result mean?** If your total score is **50 or more**, then your project **has faced serious risks**! It is very recommended to start working on these risks for which you have given the highest point as soon as possible. If the resulting value is **less than 50**, then there are no serious risks for the project yet. Nevertheless, you should definitely give thought to the implementation of the practices listed in this document.

dzmitry_harachka ZMICER.CONSULTING zmicer@jazzteam.org

4. **If desired, you can use the table for the automatic calculation of points.** For your convenience, I **prepared a table** for the automatic calculation of points corresponding to each item. There is also a separate point-value column in the table for scoring your project. Here, you can select the required point value from the drop-down list. Please, keep in mind that risks affect the final result in different ways, as each risk has a different maximum score.
   Feel free to use this table! You can also calculate the total score yourself without this table, focusing on the highest point for each risk.

   How to use the table?
   This table is available in a read-only mode.
   - If you use a Google account, you can open the spreadsheet, select the menu item **File > Make a copy**. Then you will be able to calculate the points in your copy of the spreadsheet.
   - If you do not use a Google account, you can download the table to your device. To do this, please select the menu item **File> Download> Microsoft Excel**.

Psychological signs

1. You often hear from your developers: “It's difficult, it's impossible. It will take years" [max: 3]

While you communicate with consultants, managers or other employees, and they (like you) think differently. They say that the things you ask are quite logical and reasonable.

You will find recommendations on all psychological difficulties on the project after the 4th question.

2. When you ask "Why is it difficult to implement?" developers always have some excuse [max: 3]

- «An important part of the software is unstable»;
- «A lot of code, components are too connected, everything is confusing»;
- «The manager (or other team member) won’t let me do it»;
- «We have a lot of important work before the release, we are busy.»

3. Your developers are passive and not interested in the project [max: 3]

They never come up with new ideas and do not discuss technical product improvements with you.

4. You know that one of your engineers implemented a cool improvement, brought real value to the project, but it was never applied [max: 3]

For example, a year ago, one of your developers proposed an interesting automation approach and implemented it, but it isn't still used in the work.

What are the causes and consequences (applicable to the first four questions)?

If you constantly hear such objections, see that the team does not burn with enthusiasm related to the project, when talking with you engineers look down and avoid communication in every way in general, all these are signs of learned helplessness! Such attitudes give rise to serious problems on the project: employees have no motivation, they perceive each new task more as a punishment than as a challenge. They are not productive, they cannot develop your product and offer new solutions. One can say that you are wasting resources and your project is stuck at one point.

What to do (applicable to the first four questions)?

Often, learned helplessness means that there are grievances and lack of understanding among team members. You can analyze the following issues:

1. Do engineers have the opportunity to speak openly with you and the manager about all arising difficulties?
2. Does your company have separate streams for working on technical debt?
3. Do you conduct retrospectives with the team? If yes, do you track that the decisions made in the retrospective have been implemented?
4. Can you assume that any team member can disagree with you as the founder and you will change your point of view?

Once you've honestly answered these questions for yourself, you can start implementing a regular practice of retrospectives, 1-on-1 conversations with employees, and initiate work on technical debt.

Believe me, the open, honest dialogue can solve any relationship problems the team may face!

Technological signs

To assess the technological state of your project, consider whether there are risks associated with each of the processes listed below.

Technical debts

5. There is no established practice of regular processing of technical debts [max: 5]

What are the causes and consequences?

This is a huge problem for many IT companies. The absence of the regular work on the technical debt means the absence of investment in the product's future. Of course, as a founder, you constantly try to strike the right balance between production and technical improvement of

the product. But I saw many projects that reached the point of no return. Companies just couldn't grow either technologically or in terms of quality improvement. And on each of these projects there were problems with technical debt.

And I also saw examples where well-conducted research opened up new horizons for projects. When companies could go through a difficult phase and started to create new products and to transfer experience faster.

By discouraging work with technical debt, you doom your team to learned helplessness. Over time, the code and architecture of the product become obsolete. Engineers stop believing in changes, and you, as the owner, do not see the situation on the project realistically. As a result, when you will find serious technical problems (with deliveries, bugs, lack of team trust), you will no longer be able to solve this issue on your own.

What to do?

1. Technical debts can be diverse: for example, non-implemented CI/CD and Test Automation processes, insufficient code coverage with Unit tests. You need to understand what the real problem is, what's missing on the project. **You will find a separate item for each debt in this checklist describing how to act in a particular situation.**
2. 15-20% of developers' time should be allocated to work with technical debt. This activity should become a mandatory part of sprints (please keep it in mind when planning your iterations).
3. Appeal to the team to be proactive on technical debt issues and support their ideas for process and product improvements.

Integration and delivery

6. Your project build cannot be executed using a single console command [max: 10]

What are the causes and consequences?

If you find that you cannot execute a product build using a single console command "from and to" *(including very complex projects that require the generation of keys, licenses)* and understand that there are manual operations in this process, then you're in big trouble. When you want to put a specific version of the product on a server, you will definitely not be able to just copy the build result. The team will have to carry out a number of additional activities manually. This indicates technological stagnation: the company has no automation culture developed. Your project is really at risk! You waste precious time on manual operations. If you have no autobuild, the path to automation and CI/CD implementation will be closed for the project. The same applies to almost a half of the best practices in this checklist. And unfortunately, you will not be able to compete with other companies in the market soon.

What to do?

1. Start the activity aimed at your product build process visualization. To do this, you need to create a document that will record all the actions that your team performs to prepare the build (both manual and automated). Highlight in green all automated actions, in yellow - everything that is not automated.
2. Dedicate 20-30% of your iterations time to automate the manual steps in a given process. After 2-3 iterations you will see the first results.

3. Track the team reaction. It is very important that your specialists understand the importance of this stream. If the team has supported this activity and works on it, everything is fine. If you see that the team has a negative attitude to such changes, please contact me for advice or involve professional DevOps in this process.

7. Founders (you) are not willing to invest in the process automation of the product delivery [max: 3]

What are the causes and consequences?

This is a serious issue of values, not only for you, but for the entire company. If you do not invest in important technical processes, your product has already fallen behind the competition. In addition, you lose the opportunities to reduce the cost of the delivery process, create a full-fledged CI/CD, thereby hindering the successful and safe technological future of your product. Even if now it seems that everything is fine on your project, sooner or later *(for example, when scaling up)* this issue will become acute and extremely important.

What to do?

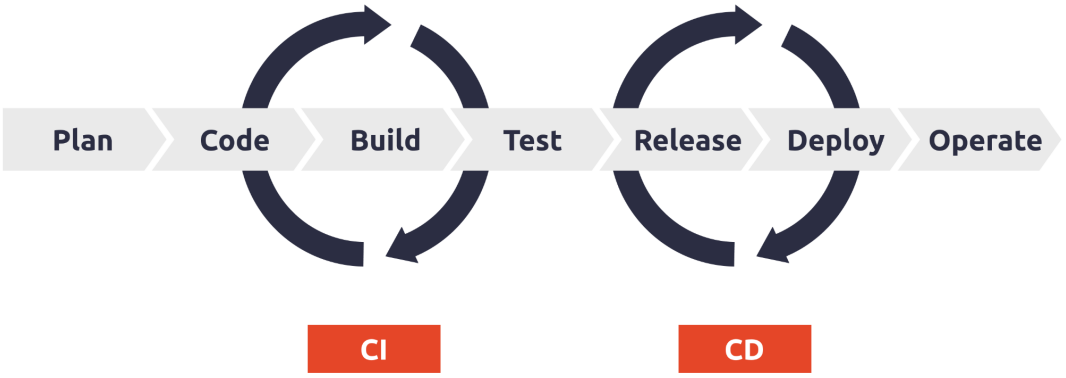
1. Start with the simple - figure out the features of this process. Part of the culture of an advanced founder is to delve into technical processes, discuss these issues with the team.
2. Once you have a basic knowledge of automation, find out if your project has an automated build: a single console command that fully builds your product. If not, this is a very bad sign (see question 6).
3. As a founder, you should clearly understand what automated deployment of your project means and how much investment it will require. Find out how many console commands you need to execute to automatically deploy your product. If the engineers answer that this can be done in a week, feel free to invest in this process. This will help to prepare for CI/CD implementation (see question 8).

8. There is no full-fledged CI/CD on the project, or not all stages of this process are automated [max: 5]

What are the causes and consequences?

Most likely, the company does not have a CI/CD culture bearer. Unfortunately, this may also be your fault. Pay special attention to the lead developer or the person who makes the main technical decisions on the project: if the question of automation didn't arise earlier, this is his big omission.

The longer you delay implementing CI/CD, the more the bus factor increases, that is, the unacceptable dependency on specific team members. Lack of automation slows down testing processes, for example, you cannot increase the number of product installations for manual (including by developers) or automated testing if necessary. This hinders business scaling.



What to do?

1. Prepare for CI/CD implementation:
    a. Be honest about whether the team has time allocated to work with the technical debt in each sprint. If not, this needs to be corrected (see question 5).
    b. Read my JazzTeam company's articles on implementing CI/CD for a deeper understanding of the topic. Also, please pay attention to the case on how to implement CI/CD step by step on a complex project.
    c. Together with developers, prepare a description of the deployment process that will be understandable and accessible to the entire team.
    d. Analyze how each part of the deployment process can be automated. If you find out that you cannot build your project with a single console command, then this is a very bad sign (see question 6).
    e. Based on your analysis, prepare a plan that outlines the steps you need to take to automate the build process for your product.
2. Start step by step implementation. If the team does not support your idea, or you encounter difficulties in the previous paragraphs, you can contact me for advice on a comprehensive implementation of CI/CD.

9. Delivery deadlines are missed on a regular basis [max: 3]

What are the causes and consequences?

If your team constantly works in the struggle mode for timely release, this indicates that something is fundamentally wrong in the project development. The result of misprioritization is "chronic illnesses" in the form of unmanageable technical debts. Most often they arise due to the lack of Unit tests, CI/CD, Data Driven Testing approach (where necessary). Of course, constant nervousness and tense atmosphere do not contribute to creativity in the team. This state of affairs will prevent the business from scaling up and working with larger customers.

What to do?

1. Conduct a retrospective with the team. Try to honestly discuss and understand the reasons for the failure of releases. The analysis of metrics and answers to the following questions will help you:
    a. Perhaps your system is very complex, while the processes are not automated and not debugged?
    b. Is everything OK with regression testing on your project?
    c. Is the project management system set up correctly (for example, Jira)?

d. Are there any signs that a DDT approach needs to be implemented as soon as possible: for example, you try to fix the same thing, but can't get control over the stability of that part of the system?
e. Do developers constantly “break” the application with their commits? This, by the way, indicates a poorly configured Git flow (see question 10).

If you answered yes to any of these questions, you should start working on technical debts (see question 5). This will help you solve the delivery problem.

Code writing and testing by developers

10. There is no conscious, well-formulated and written strategy for working with various branches of the application source code. Plus, there is no understanding of how to organize the source code of different versions of the product. The process of creating new features is uncontrollable. All this means that the project does not have a well-developed Git flow [max: 3]

What are the causes and consequences?

This primarily indicates that your technological leaders are likely to have shortcomings of culture. The consequences of such an approach are usually tragic for the project. You will not be able to develop the CI/CD culture in the development team, you will not get all the benefits that test automation provides. Also, you will be unable to isolate the work of engineers in separate branches, as a result of which they will constantly "break" the master branch. The lack of order in this matter significantly adds nervousness and provokes the risk of constant occurrence of bugs. Thus, issues that have not been resolved for years arise in the team.

What to do?

1. Ask the team a simple question: how are things with the Git flow on your project? For ease of understanding, ask to explain it in very simple terms and, if necessary, visualize the process.
2. Find out if the team has a Git flow manual that allows each engineer to understand what and how he should do in the process of preparing features.
3. Explore different types of Git flow organization with your tech leads and manager. Choose the best option for your project. My top recommendations:
   a. Always keep the master code branch up and running.
   b. Be sure to review the code before merging into the master branch.
   c. Control that your developers run automated tests and check their work in an isolated environment (see question 13) before merging into the master branch.
   d. Make sure your team has a clear understanding of which branch contains the results of each task, that all team members understand what it means to “merge branches”.
4. After the merge, all automated tests should always be run. To do this, you need to develop a CI/CD culture. In fact, Git flow ensures technological merging of CI/CD stages.

11. No Unit tests, automated tests for:
a) Server part of the system [max: 3]
b) UI part of the system [max: 4]

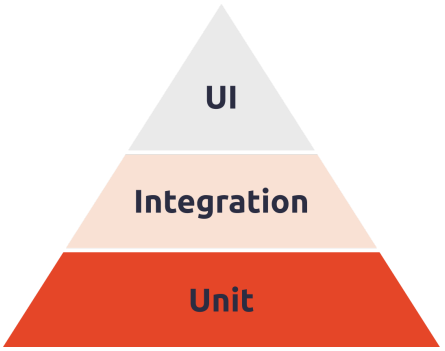
What are the causes and consequences?

Automation provides regression testing on the project, speeds up the delivery process, saves resources on manual testing. This is the standard of modern development. If you do not follow it, a very large number of relevant concepts (such as CI/CD, containerization, microservice architecture) become simply inaccessible to you. In this case, it is very likely that the company will not be able to continue technological development. It will also be impossible to safely refactor the product, introduce new components into the system, and implement new ideas.

Rejecting automation increases the cost of development, as you will not be able to hire engineers of different qualifications. Replacing and immersing new employees become very complex and lengthy processes.

Against this background, the team soon falls into a state of learned helplessness. In the development of new features, engineers follow the well-trodden path of the old architecture, without the opportunity to apply a new library or approach. At the slightest change, the application starts to behave unstable. If the amount of code is rather large (5000+ classes), the consequences are more crucial.

Without automated tests, your project is at higher risks (this issue concerns ⅔ of all items in this checklist).



What to do?

1. Analyze what kind of automation your project needs: for UI, backend or integration? Perhaps all of the above? If it is difficult to do it yourself, you can order my consultation on the comprehensive implementation of automation.
2. Figure out the current status of the culture in your company: find out if specialists have a craving for Test Automation? If engineers do not write Unit tests and do not want to start, this is a very bad sign. You need to join at least one developer who has these values to the team. Another option (more radical approach) is to set a strict rule: “New code is covered with Unit tests”.
3. Take the task under separate control: set specific time points at which the analysis of code coverage with Unit tests will be carried out, constantly make sure that there is an increase in the indicators.
4. If your project does not have CI/CD, it is recommended to think about its implementation as soon as possible (see question 8).
5. Hire Test Automation engineers for your project team.

12. There are no Unit test code coverage metrics that are understandable and accessible to the entire team [max: 2]

What are the causes and consequences?

It can be very difficult for you to track this stream without clear metrics, or you can forget about it at all. In this case, you show the team that this is not so important. By losing control over the process, you allow the same optional attitude of engineers towards Unit tests. As a result, this often leads to system instability in production.

What to do?

1. Even if the whole work of your team is not divided into clear iterations, you can implement this approach at least in Unit testing.
2. Choose how you'd like to display the metrics. This can be the number of Unit tests, the percentage of code coverage by tests. It is necessary to agree with the team how you will record this information. For quick assessment of the metrics status, you can use graphs or tables.
3. Be sure to schedule weekly meetings with the project manager and your team where this question will be discussed. Focus the team's attention on the metrics and constantly address team members with words of encouragement to write more Unit tests. If you start to track this metric yourself, the team will understand that you take the issue seriously. This will also encourage you to invest in working with technical debts (see question 5).

13. Your developers don't test their changes on a separate environment before they merge their code into the common branch [max: 3]

What are the causes and consequences?

This state of affairs speaks of clear problems with the responsibility of engineers in the company. The culture of code testing by the developer is an essential part of modern processes. If developers deny such a need, it gives rise to infantilism. They are not responsible for their code. For you, as a founder, this means too much testing cost. It also increases the iteration time. If developers don't test their code in a separate environment, it may seem that CI/CD isn't needed to be implemented.

Even new engineers you'll hire will ultimately be more expensive. Following the culture already established in the development team, they as other developers won't test their changes before merging too, so their work will have to be double-checked.

What to do?

1. Talk with the team and directly ask the developers why they don't test their code before merging.
2. Find out the developers' time estimation - how long it takes to start doing it.
3. Invest. Even if there is no containerization on your project yet, buy more servers. This is necessary in order that each developer may put their changes on a separate server (while 2-3 more servers should remain free).
4. Set a certain coding algorithm on the project.
    a. Your Git flow (see question 10) should be set up so that each developer may work on each feature on a separate branch.

b. When the developer completes work on this feature, he should locally merge his code with the code of previous tasks he has already done.
c. The developer should test his changes locally and execute the build in his branch.
d. After testing the operation of the implemented functionality, checking the completion of Unit tests and automated tests, the engineer needs to deploy these changes onto the development server.
e. The developer should check again that all changes were implemented, and the system works correctly. If bugs are found, they should be fixed.

Only after all the above actions, the code review and merge into the main branch can be carried out.

14. Your developers do not run automated tests before merging with the main branch (locally or separately on a dedicated server) [max: 4]

What are the causes and consequences?

Automated tests are not just about checking the stability of your code branches. By writing automated tests, developers can find bugs early on their own, before merging their changes to the main code **(and before these changes adversely affect the work of other engineers and the entire system)**. In addition to the obvious benefits for product stability, this develops the engineers' responsibility. It is also a culture matter of the development team and an important element of regression testing.

What to do?

1. Start by having the developers run the tests locally (see question 13).
2. Make sure you don't have commented out tests.
3. Make sure that the tests are well-written, they can be run both locally and on the server.
4. Make sure that it is easy for developers to run tests, that this activity does not presuppose a lot of effort and wasted time. As for engineer's labor, it should be like clicking a single button. It is the developed CI/CD culture that makes it possible to reduce the cost and simplify this activity.
5. Make sure your Definition of Done (checklist for particular type of tasks completion) clearly states when exactly you need to create a Unit test. If you do not use such checklists, please start implementing them as soon as possible.

Testing

15. The core of your product is not covered by Unit tests, while the capacity of multiple input values is really high [max: 7]

What are the causes and consequences?

If the capacity of input values for your system is very large (many cases, redundant situations, conditions), then I think you just cannot do without the Data Driven Testing approach. Specially if the work of a certain part of the system (kernel, data transformation logic) is based on it. Sooner or later, you will face the fact that the number of different combinations and situations will be too large, and the team will not be able to cover them manually.

Implementing DDT will not cost much as compared to the damage that can be done to the project without using this approach. Most likely, you will face reproducing bugs on the customer's side. Very often I see this situation in the cores of high technology products, complex stored procedures that implement the transformation logic. By the way, if the procedures are quite complex, the likelihood that your customer will have a rare situation in production is very high.

If you have a General Data Protection Regulation (GDPR) signed with your customer, you simply won't get access to the real data, and you won't be able to create tests for it. In the case that there are no Unit tests on your project at all, the most likely the following situation arises: you fix a bug for one data set without thinking about the same case for another data set, and everything repeats again. Or the first bug fixing can result in a new bug on a slightly different data set.

Without the Data Driven Testing approach, you will to deal with bugs in production on a regular basis. I saw examples of projects where this situation continued for years.

What to do?

1. Learn the manual on applying the Data Driven Testing approach. It contains 7 types of projects. Among them you can find your case. The article describes how to act in each situation. You can also study my company case on the successful implementation of the DDT approach on a complex project.
2. Launch a separate technological stream to implement the DDT approach on your project. Of course, if you do not have CI/CD and test automation implemented, you should start with them.

Contact me, and I will help to implement DDT within zmicer.consulting service.

16. There are no documented test cases for manual testing [max: 3]

What are the causes and consequences?

This means that your project does not have a culture of test management, testing is not systematic and uncontrolled. In this case, you cannot completely track the quality of the product, understand how well the testing is done, and also seamlessly replace people in the team. Moreover, if you don't use professional tools for test case management, it limits your best practices. And when the load increases or the product is actively used, you will definitely have quality problems.

What to do?

1. Introduce the documentation practice. Let employees who are responsible for the project quality start writing test cases.

2. Assess the team's level of resistance in this matter.
3. Implement and configure Jira for work with bugs. All team members should understand how to record bugs, which versions of the product these bugs belong to.
4. Start introducing manual testing into your delivery cycle.

17. Special test plan management tools are not used (TestLink, TestRail) [max: 3]

What are the causes and consequences?

At the first stages, it is acceptable to create test cases in separate documents, Confluence, google-excel spreadsheets. But if the capacity of your product is large and constantly increases, it is recommended to introduce professional test plan management tools as soon as possible. This will allow you to accumulate the history of checks, record regression, visualize the percentage of code coverage with tests for different domains, analyze which part of your functionality is the most unstable, thereby it will influence Test Automation introduction. And one more important aspect: using professional tools, you can understand how long it takes to complete the entire test plan, and clearly understand how many people are needed to carry out this activity.

TestRail

TestLink

What to do?

Start using these tools. They significantly reduce the time and cost of manual testing. In the future, it will be much easier for you to integrate the testing process into CI/CD.

18. No connection between test case sets for manual and automated testing [max: 1]

What are the causes and consequences?

This situation is often observed on those projects where the CI/CD process is not set up, and there is also no competent test management. Manual testing follows some principles, while there is no automation at all, or it is implemented, but it is impossible to track what part of the cases is automated. In such cases, it is impossible to clearly assess the level of code coverage by automated tests, to understand which part of the system is more stable and which one contains most bugs (“bottleneck”). Basically, you cannot invest in Test Automation in a manageable way. If you have a clear link between manual and automated test cases, you can control the number of team members (manual and test automation engineers) that you need to ensure the stable operation of the system.

What to do?

1. Ask your QA engineers to write down existing relationships between sets of test cases (if the project uses professional test plan management tools, this can be done automatically).
2. Organize a meeting of manual and automated testing teams. At the meeting, specialists should decide which of the existing cases for manual testing should be automated.

3. Outline a plan for increasing code coverage with automated tests. Display this metric with graphs and track the progress regularly.
4. On a regular basis make sure that the share of automated tests increases. If this does not happen, explicitly ask the team what the reason is. Especially if you periodically observe unstable system behavior.

19. Unforeseen bugs in production that emerge on a regular basis. You feel that you cannot stabilize some part of the product by any means [max: 2]

What are the causes and consequences?

The systemic occurrence of unforeseen bugs in production is just the tip of the iceberg. And this suggests that the product needs to be improved. The following issues can be the source of this problem:

1. Problems with the test plan.
2. Problems with regression testing.
3. The presence of a "bottleneck" - a part of the system that is not covered with automated tests.
4. Wrong merging process, incorrect work with code branches.
5. Maybe your engineers don't test their changes before merge (see question 14).

What to do?

1. Implement regression testing of emerging bugs.
2. Structure each unexpected bug by cause and by type. Find answers to the questions: what this bug is, why did it occur?
3. All bugs should be visualized and included into reports.
4. Group bugs by the part they occurred. Perhaps the bugs are concentrated in one part, which causes a bottleneck. In this case, you need to apply the Data Driven Testing approach (see question 15).

Immersion of new team members in the project

20. There is no clear textual description of the architecture, as well as the key features and processes of your project [max: 3]

What are the causes and consequences?

In this situation, there are several serious risks at once. First, the company develops a high dependence on one or more key employees. If this employee quits, it will be a huge stress for the entire project. Team members will have to puzzle out the architecture anew. If you don't have a clear product description, it comes at a high cost to bring new engineers into the project.

Secondly, it can speak of colossal value problems with developers. Without knowing the specifics of the architecture, developers will be afraid to evolve the system core, change the architecture to comply with current technologies. And all development will slide into bug fixing and creation of new features purely based on current developments. I saw projects where engineers had been "hooking on support" for years, and could do nothing substantial. Having a team that cannot move the project forward is a very big risk that prevents business from scaling up. And it’s very likely that when larger customers come to you with serious requirements for architecture (containerization, performance), you will not be able to offer them anything.

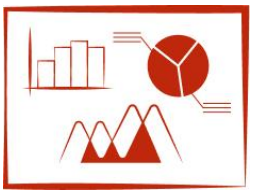
What to do?

1. Start activities to document the architecture and features of the product. To do this, you need to choose an experienced engineer who can simply and clearly describe the architecture. Agree that he devotes 1-2 hours to this task every day. I think it is absolutely possible to complete this activity within a week.
2. Prepare a video in which your lead developers tell about the product architecture in plain language.
3. Add an item to the list of the managers' responsibilities that the description of the product architecture should always be kept up to date.
4. Prepare a list of product improvements, discuss with the team what they would change in the architecture.
5. Think of a Newcomer Pack for new employees that will help them quickly dive into your codebase.
6. Consider preparing a reference project for your product architecture (see question 21).

21. Your system has a large codebase, while there is no playground (reference project) where novice engineers can dive into the project [max: 2]

What are the causes and consequences?

The reference project contains all the main libraries, approaches and architectural solutions that are used on the main project. Typically, a reference project is implemented on a small model and allows new employees to quickly immerse themselves in the features of the architecture. **Based on my experience, I can say that the use of playground speeds up the immersion of a new engineer by 5-7 times.**



Why is it so important? Sometimes the combination of technologies is so complex and intimidating that novice engineers are simply afraid to dive deep into a project. This leads to the fact that those developers who have been working on the project for a long time know everything, and the newcomers do not understand the features of the system architecture. Unbalanced relationships within the team do not contribute to the system development.

If you do not have a reference project, too much time will be spent on immersing new employees, up to several months.

What to do?

1. Study the case describing how to accelerate the immersion of engineers in the project.
2. With the help of your developers, make a list of key technologies and architectural approaches that are used on the project. Do not include libraries in this list that are not conceptually important and have no impact on your project.
3. Brainstorm with developers. The purpose of the meeting is to come up with a very simple case that is as close to your system as possible, which requires writing a small amount of code. You do not need to completely reproduce the architecture of the system. But at the same time, it is important to make sure that the core technology stack of your project is involved in the playground. **My recommendation: the volume of the task aimed at creating a reference project should allow a new engineer to complete it within a week, even better - in 3 working days.**

4. Catch the moment when a new employee comes to the team. His first task should be to create a reference project based on the prepared case. At the same time, it is important that your team actively help the newcomer, answer the questions, provide the necessary information and documentation. The result of the work should be not only written code, but also documentation for the system deployment. In addition, the new developer needs to record which of the assigned tasks he managed or did not manage to implement. Later, you will be able to assign uncompleted use cases to new generations of engineers coming to this project.
5. As soon as the reference project is ready, it should be actively used to immerse each new developer (they will thereby improve your playground).

Problems with management and processes

22. Your manager is not interested in and doesn't understand the technical side of the project; CI/CD and control of Unit tests mean nothing to him [max: 5]

What are the causes and consequences?

The popularity of the Agile approach over the last 10-15 years has dramatically changed the requirements for project managers. A modern project manager should understand the basics of CI/CD, automation, and basically everything covered in this document.

Why is that? If your manager does not understand technical issues, he is a kind of sit there: de jure he performs management, but de facto he doesn't delve into the essence. There is a high probability that he makes wrong decisions that will negatively affect your product in the future. Moreover, if the team is at the stage of learned helplessness and there are no obvious technical leaders among the employees, then in fact your project is a system of spontaneous accumulation of technical debts and negativity.

The manager who does not make competent decisions, including in technological areas, dooms the project to failure.

What to do?

1. The problem should be discussed directly with the manager. Explain why it is so important to understand the concepts and the technical side of the product. My management standard implies that the manager boldly enters the technical side. This allows him to communicate with developers in the same language, respectively, to have more trust on their part. This also is necessary to initiate technical improvements, manage the processes of working with technical debts.
2. Outline a manager training plan. In a short time, he must obtain the necessary minimum of technological literacy, understand the CI/CD and Test Automation processes. He also needs to learn how to create deployment diagrams, components, in order to understand the principles of your software at the data movement level. My experience in creating a management office and training managers shows that it is not a problem to master all this.

23. No well-tuned project management system [max: 2]

What are the causes and consequences?

In fact, most companies have a project management system, but, unfortunately, do not advance to its professional applying. The system is simply not set up to such extent that the team can use it conveniently (accordingly, the team does not see any point in this).

Properly applying the system of filters and Kanban boards, you and your team could get a complete and understandable picture of the project status with a single click. It should be easy for you as a founder to understand what is in progress, whether everything is fine with deadlines and estimations. If such activity takes too much time, the project management processes are obviously incorrectly configured.

What to do?

Invest time in setting up Jira properly.

If you have any problem to do it yourself, feel free to contact me for such a service.

24. The system of task estimation and decomposition is not implemented [max: 2]

What are the causes and consequences?

If your specialists don't estimate and decompose tasks, they won't be able to realistically estimate the necessary costs for different activities. For you, this means low control over engineers and lack of transparency. Perhaps, you spend too many resources on their work. Some tasks may be in progress for weeks. This approach can lead to dependency, infantilism of developers.

What to do?

1. Start by introducing a culture of task decomposition. Establish the following rule on the project: after decomposition, any task must be completed within one working day. Be prepared for the possible team resistance. Engineers may insist that they already carry out decomposition, but if the task is too complex, sometimes work on it can take a whole week. Explain to the engineers that this is a wrong approach. **The result of any task that's being performed should be measurable, clear and demonstrable by the end of the day.** If developers are not required to be serious about such an important issue, this can lead to excessive slackness and ineffectiveness. Of course, if a team of Senior engineers is working on your product and there are no problems on the project, then you can don't raise this issue (but probably, then you would not have read this checklist). In general, I always recommend introducing the practice of task decomposition in the company. This contributes to the progress and promotion of responsibility in the team, improves logistics in advancing the project tasks.
2. Gradually introduce the developers' time distribution system: **3 small tasks of 2-3 hours** must be completed per day. At the same time, the result must be measurable,

concrete and demonstrable. For example, if your developer is learning a new technology, ask him to prepare a manual or diagram demonstrating how to use it.

3. At stand-ups, engineers should analyze and explain to the whole team why they didn't manage to complete the task on time. All project participants should strive for realism, constant progress, and understand that completing a task within the stated estimate is the norm.
4. If an engineer is engaged in a research task, the following conditions should be set: **1-2 hours can be spent on this task every day for several weeks**. **After that, the assessment of the obtained results should be performed.** In my experience, the best progress is achieved by doing the task gradually. In this mode, the brain has time to delve into the question posed, and creative energy gets involved. Good ideas can come even while sleeping or taking a shower.

25. No description of project benefits for the team [max: 1]

What are the causes and consequences?

Every project has interesting challenges and tasks. Unfortunately, over time, this information is forgotten, engineers can lose some of their enthusiasm. The formulated description of the project with all its advantages and features helps to attract and retain specialists, increase their involvement in the project.

What to do?

Do internal marketing and actively promote the brand of the project within the team. To do this, you need to create and regularly update a document that will describe all the features and bonuses of the project, the benefits, prospects and opportunities that engineers get by working with you. You can also record a video about the project or prepare a presentation.

26. Your team works remotely, but does not turn on cameras at stand-ups [max: 1]

What are the causes and consequences?

In my consulting practice, I often encountered distributed development teams where engineers didn't turn on cameras for years. With such communication, employees develop a formalistic attitude towards the project, and a sincere human connection is lost. Voice communication completely turns off the non-verbal communication channel. This allows the team to be distracted and not to try hard, not to give energy and not to focus on the topics that are discussed at the meetings.

What to do?

Start with yourself. More often turn on the camera when chatting. Also, gradually introduce this rule into the team over time.

27. You don't attend demos and stand-ups in person [max: 2]

What are the causes and consequences?

It can be said for sure that you don't leverage to the full extent. I often face situations when an owner without a technical background does not immerse himself in the project and does not want to participate in meetings with the team. But this is very important in order to control the project progress, to understand the real causes of difficulties and problems.

Over recent years, the development culture has improved a lot. What earlier was considered to be just technicalities now is the standard which everyone involved in the project should understand.

I constantly advise business owners, and can say that even the founder without a technical background can become well-versed in technical issues in 6-12 months. And the team will definitely notice it.

What to do?

1. Start figuring out the CI/CD process and the impact of Unit tests. Clarify who decides on project releases, what DDT is, why retrospectives are needed, to what extent your project is represented in Jira.
2. Having gained basic knowledge, you can start sometimes coming to stand-ups and demos with the team, gradually immerse yourself in the project.
3. Create a culture of trust on the project, when employees are not afraid to openly say that they do not understand something. And, of course, be an example for them: show interest, communicate with engineers, ask them questions. So you will be aware of the real situation on the project and in case of difficulties, you will be able to prevent serious consequences.

HAVE ANY QUESTIONS?

You can contact me for a free consultation

Zmicer [Dzmitry] Harachka

dzmitry_harachka

zmicer@zmicer.consulting

zmicer_consulting