2018

Measuring Engineering

A REPORT

NIKITA USHAKOV

Introduction:

Before providing a report on the software-engineering process that has emerged because of the accumulation of experience during my studying and internships in recent years, I would like to make a few general explanations that seem to me to be the most significant.

I worked in the IT industry only this year, although I began to engage in object-oriented programming much earlier, about 6 years ago. My main activity during my experience as a system architect was the optimization of the work of algorithms, the development of machine learning algorithms and high-level architecture, and control over the implementation of the concept throughout the project. In addition to participating in the development of software and writing code. Also, I spend some time on my own projects at the development of game modifications and their support, where not only knowledge of the language and development environment is necessary, but also their knowledge of the structure of the game engine itself, which can present unpleasant surprises.

The projects I worked on the most are most often associated with the development of open-source software. But unfortunately, I didn't have significant experience in working with developing software and programs focused on the release of a full-fledged investment product.

However, I know that custom software can be designed for the internal or external customer. Exclusive rights to the developed system are received by the customer, and work on the development of the system can later be transferred to another performer, for example, a developer team. And such development requires more real-world experience than simple experience in open-source projects.

But it is also worth mentioning the investment software, which attracts developers even more since the work on it is carried out by the performer for the money of an internal or external investor. As a rule, the rights to the system code remain with the contractor, which stimulates continuous work to improve their product and the consistent release of versions with more advanced functionality.

The customers of such software as investment and custom software can be governments, large state-owned and commercial organizations, etc. Therefore, in the sense of custom software, there is often some difference between the product development processes for internal and external customers. I want to complain about such nuances in my analysis. First, the level of formalization of relations with the customer can vary greatly from project to project. For example, the larger the project budget, cause formality growth. For example, a government customer or large commercial enterprises (especially with state participation) usually have legislative restrictions on the formation, placement of an order and acceptance of work results. Another limitation of large organizations is the fact that their staff, which can both act as a catalyst of requirements and the main user of such products, have very limited accessibility for performers, if only because of their employment. However, for small organizations, the level of formalization falls and sometimes goes to the opposite extreme, where there is an insufficient level of responsibility for the customer within the project.

The other side of any custom projects is many strict functionality requirements. This is a high load on all systems and a large geographical distribution, and high demands on the accuracy of calculations with a very limited time frame. Usually in such projects, there are elements of research and creative search aimed at solving non-trivial design problems. Sometimes, developers will need to combine capabilities within a single development process with different methodologies. This will allow keeping a low level of user involvement, related to the nature of

the project, with the flexibility of development in conditions of high uncertainty of requirements. In this regard, it is the preparatory stage that is important for professional developers, during which it is possible to choose the necessary methodology and build an optimal development process.

As an example of work on an investment project, I can cite participation in the development of a comprehensive program called "Flexter" in the small company "Sonra". With my participation, several versions of this software were released, the users of which were various commercial and government organizations in different countries. Nowadays, this software is near 600 times faster than ETL tools. It significantly reduces any colossal data conversation. In addition, it was fully written using the functional language called “Scala” with using “Spark API” tools, which allowed the efficient and comfortable environment on working with data frames on different machines at a same time for increasing the convertation runtime.

During my internship, I understood, that the process of developing an investment project is a list of certain stages and how the companies can measure your participation. The software classification I quoted is made only to show a possible difference in the organization of the development of various software tools. Reviewing the development process, I will focus only on the differences of the process itself with regards to different types of software. However, it must be remembered that the differences between the processes of developing different types of software are much deeper. Therefore, during planning each stage, these nuances should be considered.

In addition, the transition of the process from one stage to another does not have a clear boundary. Basically, the work of the next phase begins as most part of work is completed at the previous phase, but not fully complete yet. This especially concerns the development of requirements, when in some cases the removal of uncertainty occurs only towards to the end of the project. Of course, the presence of such uncertainty in the project is a significant risk and should be under constant control.

Basics of the measuring:

Checking the correspondence between the actual and expected work of the developer, performed on the all developing stages and measurements can happen in different stages and be not equal depending on the software process type. In a broader sense, I think that taking a very cautious approach in deciding how to measure developer productivity, because of the most traditional methods, such as string codes, number of commits, number of corrected errors, appearing to be subjective for today's software development concepts. We must evaluate the approach to team performance, as well as evaluate individual key performance indicators in the project.

Look at general measurements of developers in modern companies is:

1. **Fewer mistakes - more efficient worker:** The fewer mistakes a worker makes in work, the better his work. The quality of the code easily turns into the pleasure of users who can choose your product exactly for this indicator. The second plus of quality code is to minimize the cost of support, testing, and more. Yes, the person is not perfect, but if he didn’t test the results of his work on his own but had already reported on the completion of his work, then his laziness would lead to a whole chain of ineffective gestures in the company. For the mistakes of employees, the company pays out of its own pocket, and this then reflects on the well-being of the workers themselves.

2. **The more effective the employee, the more he spreads useful information in the team:** In the development team, there should always be an information exchange. Employees must share useful information about convenient code designs or techniques, about the successful implementation of some functionality, about technologies, about the results of their research, about problems and their solutions. If a person does not generate information, he ceases to be an effective employee. as his knowledge does not help others. This does not apply to newcomers to the team, but even newcomers can create a positive information background.
3. **Idea generators are many times more efficient than consumers of ideas:** Employees who can independently generate ideas are, in my opinion, the most valuable employees. They do not need to write detailed technical tasks, they do not need to scrupulously explain the difference of one approach from another, and why it is worth choosing a different path, and not the one that they have read somewhere. These workers will come up with 10 ways to solve non-trivial tasks and will find 10 reasons why each of them will not work. They drag the whole team with them and are not afraid of difficulties. Because they are interested in reinventing bicycles, and in no case should this opportunity be taken away from them.
4. **The more abstract thinking, the more effective the employee:** The simpler abstractions a person operates, the simpler the task he can solve. Ability to manage complex abstractions leads to a tremendous effect. For such a person, the programming language, tools, approaches, algorithms cease to be important. Any, even the most complicated system, will not pose any difficulty for him, because he can look at the root, and not at the tops. Usually, such people become software architects, develop specifications, and engage in analytics.
5. **The effectiveness of the employee is directly proportional to his self-motivation:** There are two types of workers: “tired” and “aggressive”. “Tired” workers tend to do as little as possible and receive as much money as possible. They are not interested in developing or learning something new. Their fatigue is not only the fact that they have been burdened with a lot of work, or they have not efficiently distributed their time, and they have had to recycle. Their weariness from loss of interest or motivation in work. “Aggressive”, on the contrary, try as quickly as possible to go through the path of learning, development, becoming, to gain access to even more resources to develop, develop, develop. They eagerly absorb knowledge, learn, try, make mistakes, and over them there is no need to build a pyramid of managers to spur to work. "Tired" are trying to find work in large companies, where the effect of their work is not visible at all. “Aggressive” find themselves in startups, small companies, because there it’s easier to get everything they need.
6. **The faster employees perform code refinement, the more effective they are:** Because at least they provided for the scalability of the code, implemented the code itself at a qualitative level, using understandable constructions and variable names, can quickly read “someone else's” code, created a predictable and understandable architecture in which debugging is easy to do. Such a simple characteristic can tell about the code much more than the number of lines per unit of time. These are not all assessment methods, but these are the ones I use for personnel selection and for evaluating the effectiveness of a company. I hope that you will come in handy.

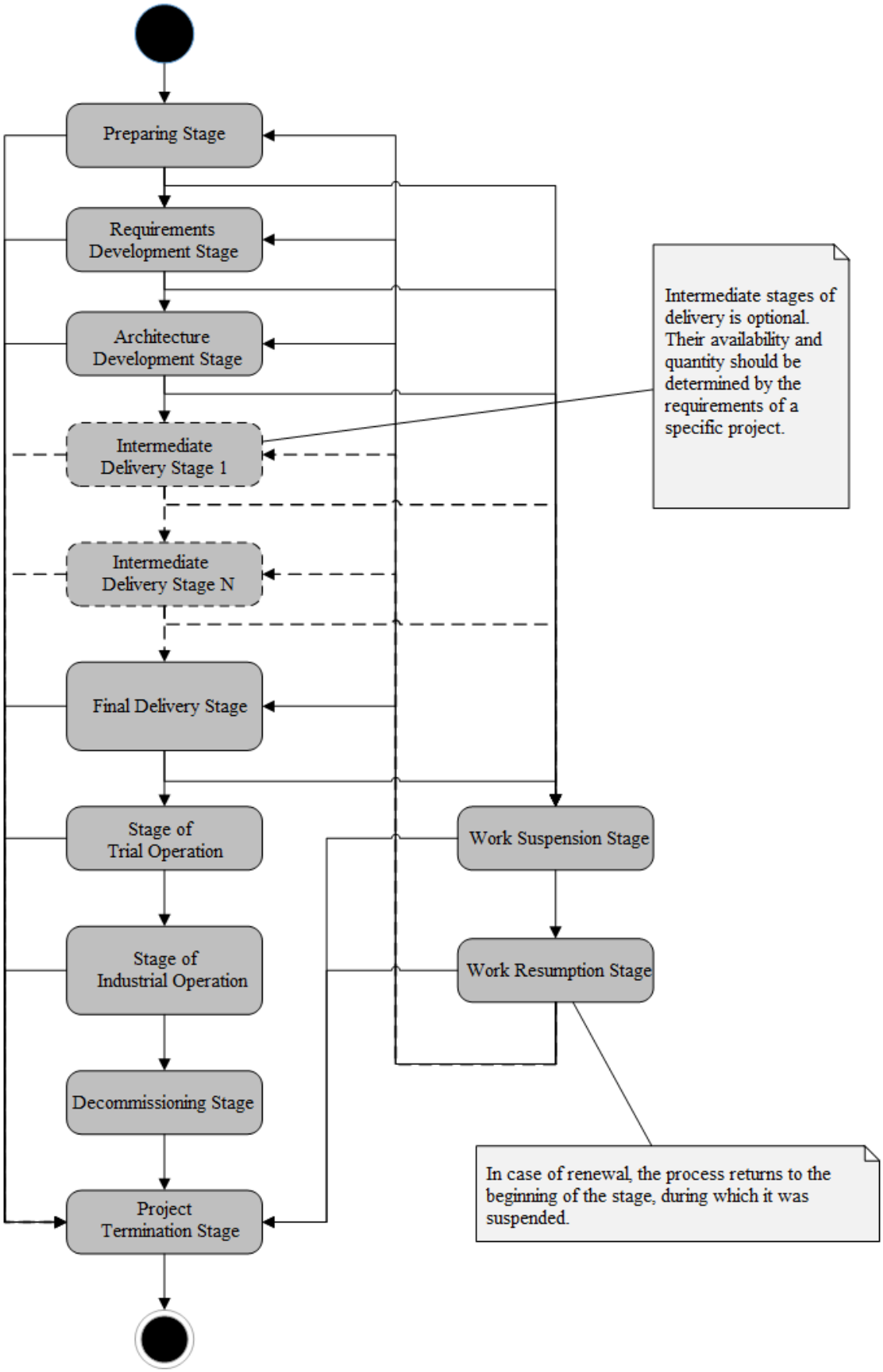
Types of data for improving the measurement is:

1. **Review of code:** In any project, developers can decide that the number of reviews of the code should be carried out within a certain period. Based on code reviews, people get comments on improving standard coding standards. Attention should be paid to the recurring questions of revising the code of the same code. You can use automatic code checks or manual code reviews.
2. **Lines of code:** A very outdated method for counting lines of code that developerы write for a certain time. Many companies that had experience in paying for the number of lines of code that the programmer wrote concluded that this is unwise, because the developers tried to create working code with a terrible amount of excess code. Money is paid for the number of characters, not for the quality or efficiency of the code. At the dawn of the development of the IT industry, there was a popular good practice called "KISS", which obliges to focus on writing the most simple, basic code available as far as possible while maintaining high quality. Although there was another metric called "LOC", which in turn encouraged distribution of the code in several lines, when it could be written as effectively and more clearly on one line. However, a decline in quality to the detriment of quantity, an increase in technological debt. Motivation, interest, satisfaction - everything is rapidly falling. As a result, routine and low qualifications.
3. **Number of commits:** A similar method of counting lines, but then counting the number of commits for a certain time limit. It will also protect the project from losing important components and saving all changes can help roll back to the previous version and eliminate errors. However, it is obvious that the developer can make useless commits for the purpose. therefore, this is not a good perspective on analyzing developer productivity.
4. **Bugs:** The higher the developer level, the more he gains experience. The more experience, the less mistakes are made, since many of them have already been passed. If a person knows how to learn from his mistakes, then the number of mistakes decreases. Over time, the developer understands that his task is not just to write code, but to help the team solve an any problem. Understanding the value to the end user, the developer begins to think when checking his own work, like this user, thereby checking the most important cases. One of the criteria of a high-class programmer is the stability of his code to change. Few people write bad code, however, a good developer, like a grandmaster in chess, is distinguished by a certain strategic vision of product development. Even creating a simple functional, it provides for the possibility of its change in the future, therefore, with the expansion of the product, there are less regression bugs.
5. **Test quality:** How developer use verification of the correspondence between the actual and expected behavior of the program, performed on the final set of tests selected in a certain way. In a broader sense, testing is one of the techniques of quality control, which includes activities on scheduling "Test Management", designing tests "Test Design", performing tests "Test Execution" and analyzing the results obtained "Test Analysis".

However, in some projects, the measurements of the development team and the expected performance are decided based on releases. With each release planning, there are different “contracts” agreed with team members for expected performance. I believe that these approaches are more successful, since there is no reason to stick to UI-related measurements in the release, where a complex algorithm will be released. But to look more detailly we should know the structure of developing different types of software and how they are planned.

Custom software development process:

Let’s begin with the development of custom software. The basic process diagram is shown below.



The work on the project begins with a preparatory stage. The purpose of the stage is to create some basic concept of the future software product based on the customer's proposals and starting from this concept to assess the relevance and feasibility of the project. If the decision on engaging a performer is made by the customer on a competitive basis, the preliminary stage is the phase of preparing a potential performer for the competition, including the formation of the necessary documentation.

The developer team should try to not spend time and resources on a project whose concept is deemed unclaimed or unrealizable. Such a project must be purged. In some situations, some iterative work with the customer is required to make some corrections of the project's concept, until either an acceptable balance of customer requirements and the contractor's costs is reached, or a decision is taken to minimize the work.

The project's concept, which looks acceptable for implementation, is entering the stage of developing requirements. At this phase, the developer team must create a list of all the obvious and hidden needs of the customer. It often turns out that the customer has either not decided on his needs, or his needs conflict with each other, with the capabilities of the customer or with the capabilities of the performer. The objectives of the stage are the identification of all hidden needs, the resolution of conflicts of requirements, the formation of a holistic technical solution and an analysis of the feasibility of the prepared solution.
However, clarification of requirements leads to a revision of the project concept. Imagine that after clarifying the process of all requirements, it is not possible to find an acceptable technical solution, the project has to be curtailed or postponed for some time, waiting for more acceptable circumstances. If a technical solution is found, the executor proceeds to develop the architecture of the future system. The goal of the stage is to define a high-level logical and physical architecture that fully covers all customer requirements. When developing the architecture, a review and clarification of the concept, requirements and a preliminary technical solution is carried out, which makes it possible to prevent the most dangerous risks.

After creating the design of the architecture, it is necessary to revise the main parameters of the project and decide whether the executor can complete the project. It is useful to abandon unnecessary and ponderous functions at the architecture development stage. Optimizing an architectural solution often helps fit into acceptable project parameters. In other cases, a more radical reduction in the functionality of the system being developed is required. However, even stopping the project at this stage, if it happens for good reasons, should be perceived as a victory, because of the continuation of work, in this case, can only lead to even greater losses.

If a balance has been found and it was possible to create an acceptable system architecture, the contractor can proceed to the implementation and delivery of the system. An implementation can take place in one or several stages. For small projects, a one-stage delivery of all the functionality of the system may be quite acceptable. However, the larger the project cause the higher the dependencies of the subsystems. Under these conditions, the implementation should be divided into several stages so that at the end of each stage the development team has a product ready for delivery. In this case, the most vital fundamental functionality should be developed in the early stages, and add-ins that work on top of these basic components should be implemented later. In this case, the most dangerous errors for the system will be corrected in the first stages, and the risk that the applied functionality of the system will be based on an unstable basis will be significantly reduced. After the delivery of a fully completed system, the custom design project usually proceeds to the trial operation phase. The purpose of this stage is to check the quality of the developed system in real conditions of operation. As a rule, at this stage, the contractor, together with the customer, takes measurements of quantitative metrics to determine the quality

of the system created. First, the functional characteristics of quality are checked, then non-functional. If there are inconsistencies, the developer team should correct the system code.

Fully debugged and configured system is introduced into industrial operation. As a rule, the contractor must accompany the system, at least during the warranty period. Detected inconsistencies should be corrected. Users and customer service staff should receive operational advice.

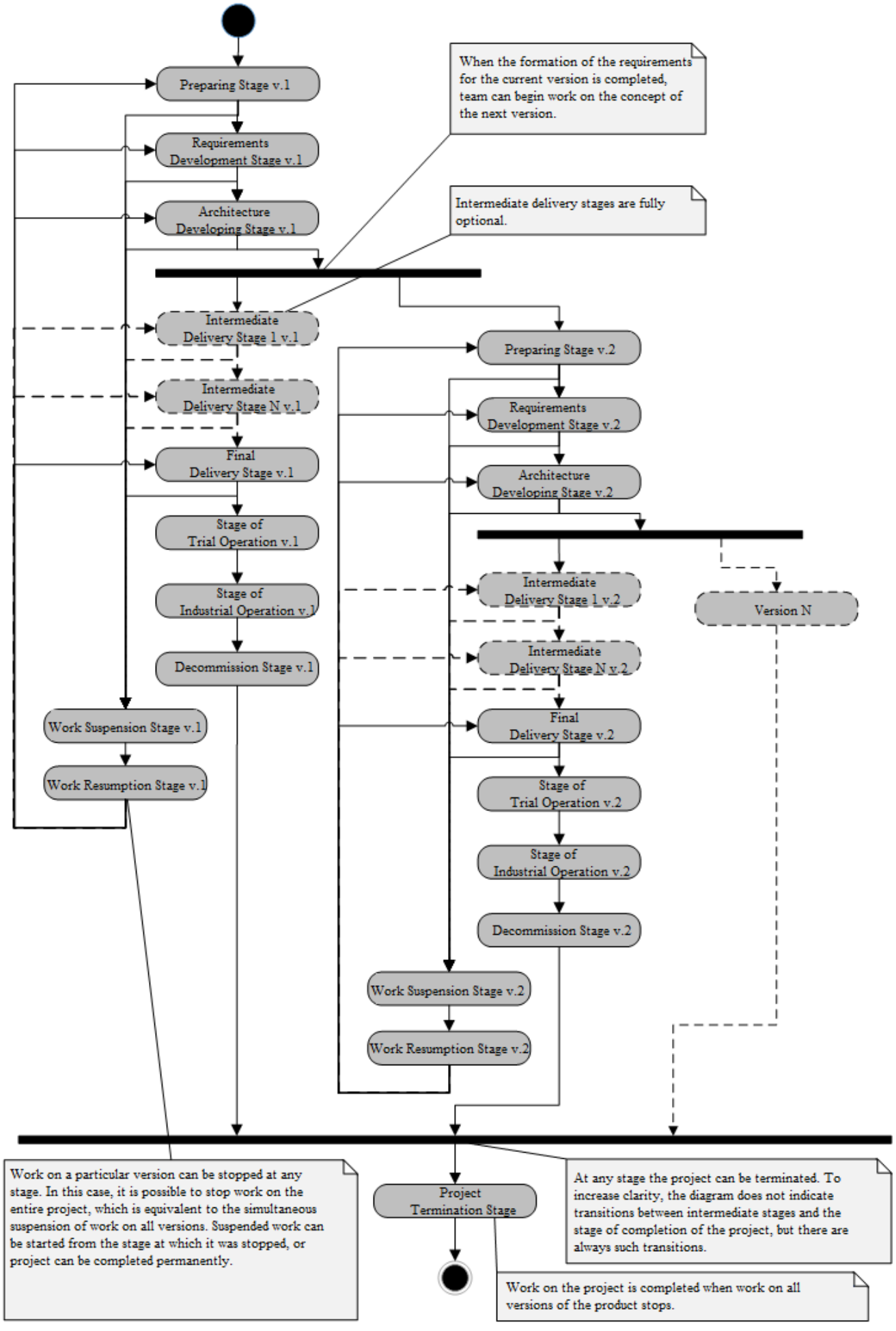
However, when the system ceases to arrange the customer for any reason. The stage of system decommissioning begins. However, for custom software, this stage is not always relevant, because the customer can use his exclusive rights to the system and remove the contractor from further work on the maintenance and development of the system even before it loses relevance.

Any project ultimately comes to an end. The project termination phase aims at analyzing the results, making changes to the development process based on the experience gained and adding new effective solutions and precautions to the developers' knowledge base, as well as new ready-made components that can be used in future projects.

Finally, I should mention another two more stages of the development process. It happens that circumstances do not allow the project to continue, but the results of the work done show that the project may have a possible future. Close this project is quite untimely. Therefore, instead of completely stopping the work, the developer may suspend the project activity, recording the results achieved. As soon as circumstances allow, the project can be resumed, having the infrastructure re-established, having returned the developers to the project and restored the state of the project. However, to resume work from the stage at which the project was interrupted, re-conducting an audit of the results achieved.

Investment software development process:

The process of developing investment software is interesting in way that the work can proceed simultaneously on several product versions: while the first version is close to release, the second is already being implemented, and for the third one the requirements are formulated. The process is shown in the diagram below.


Preparing Stage v.1
When the formation of the requirements for the current version is completed, team can begin work on the concept of the next version.
Requirements Development Stage v.1
Architecture Developing Stage v.1
Intermediate delivery stages are fully optional.
Intermediate Delivery Stage 1 v.1
Intermediate Delivery Stage N v.1
Final Delivery Stage v.1
Stage of Trial Operation v.1
Stage of Industrial Operation v.1
Decommission Stage v.1
Work Suspension Stage v.1
Work Resumption Stage v.1
Preparing Stage v.2
Requirements Development Stage v.2
Architecture Developing Stage v.2
Intermediate Delivery Stage 1 v.2
Intermediate Delivery Stage N v.2
Final Delivery Stage v.2
Stage of Trial Operation v.2
Stage of Industrial Operation v.2
Decommission Stage v.2
Work Suspension Stage v.2
Work Resumption Stage v.2
Version N
Work on a particular version can be stopped at any stage. In this case, it is possible to stop work on the entire project, which is equivalent to the simultaneous suspension of work on all versions. Suspended work can be started from the stage at which it was stopped, or project can be completed permanently.
Project Termination Stage
At any stage the project can be terminated. To increase clarity, the diagram does not indicate transitions between intermediate stages and the stage of completion of the project, but there are always such transitions.
Work on the project is completed when work on all versions of the product stops.

It clearly shows that in the development of investment software, the same steps take place that was discussed above for the process of developing custom software. But the difference is that the stages do not relate to the whole product, but to a separate version of the product. The exception is the stage of project termination: the project cannot be completed while work is in progress on at least one version of the product.

The most important part is the beginning of work on the next version of the product. This moment comes as soon as the stage of creating the architecture of the current development version is completed. Prior to this, at the stages of forming requirements and creating architecture, as a rule, there is a discussion about which functions should be implemented in the current version and which ones should be transferred to the future. And only when the requirements for the current version are formulated, reviewed and confirmed by the system architecture, does it make sense to think about the next version.

In addition, after the development of architecture, analysts and architects of the project have some leeway, because at the delivery stages the main burden falls on programmers. This freedom can be used to develop the concept and requirements for the next version.

Also, it is possible to postpone the start of work on the next version to a later date. For example, it is quite possible to first enter the current version into experimental or even commercial operation, and only after that begin work on the next version. But it must be remembered that such a decision is not applicable in the case of high competition: developers will simply be outstripped and squeezed out of the market. The decision must be made based on the whole complex of circumstances affecting current business.

Speaking about the process of developing investment software, important is to understand that working on several versions has several explicit and hidden interdependencies between the parallel branches of the process.

Firstly, corrections of inconsistencies identified in the earlier version should be made in the version where they were found, and in all later versions, including those developed versions. This applies not only to the program code but also to all other artifacts of the project, like a technical and user documentation, help system, assessments, and work plans, etc. Moreover, corrections must be made immediately, because you will not be able to reduce the cost of corrections, but if developers do not make corrections immediately, their cost at later stages may increase by tens and even hundreds of times.

Secondly, for parallel work on several versions, a special project infrastructure is needed, including organizing control of code versions and documentation, control of tasks and inconsistencies, automatic build and testing utilities, etc. Developers should prevent any work on one version of a product to block the execution of tasks on other versions only because the project infrastructure does not allow launching two assembly processes simultaneously for different versions of a product. Also, special attention should be paid to the tools on which testing is being carried out, for example, all versions of the product that were previously released and all versions that are currently being developed should be deployed.

Thirdly, the same participants can be simultaneously involved in the work on several versions. There is a big risk that a key employee may get bogged down in working on one version of the program and allow significant time overruns in tasks related to the other version.

Finally, the reverse situation occurs when the personnel working on one version do not know anything about what decisions are being made as part of work on another version. Part of the

problem is resolved if the fixes of all the documentation and code are immediately extended to the later versions I mentioned above. But the matter should not be limited to corrections. It is necessary for the team working on one version to understand why certain decisions were made when working on another version. However, a knowledge base for developers is needed - a special information system in which all problems encountered by developers when working on a version of a product, and ways to solve these problems should be described. The knowledge base should send notifications on new entries to all project participants. You cannot let the interaction of two teams working on different versions of the same product take their course.

Embedded software development process:

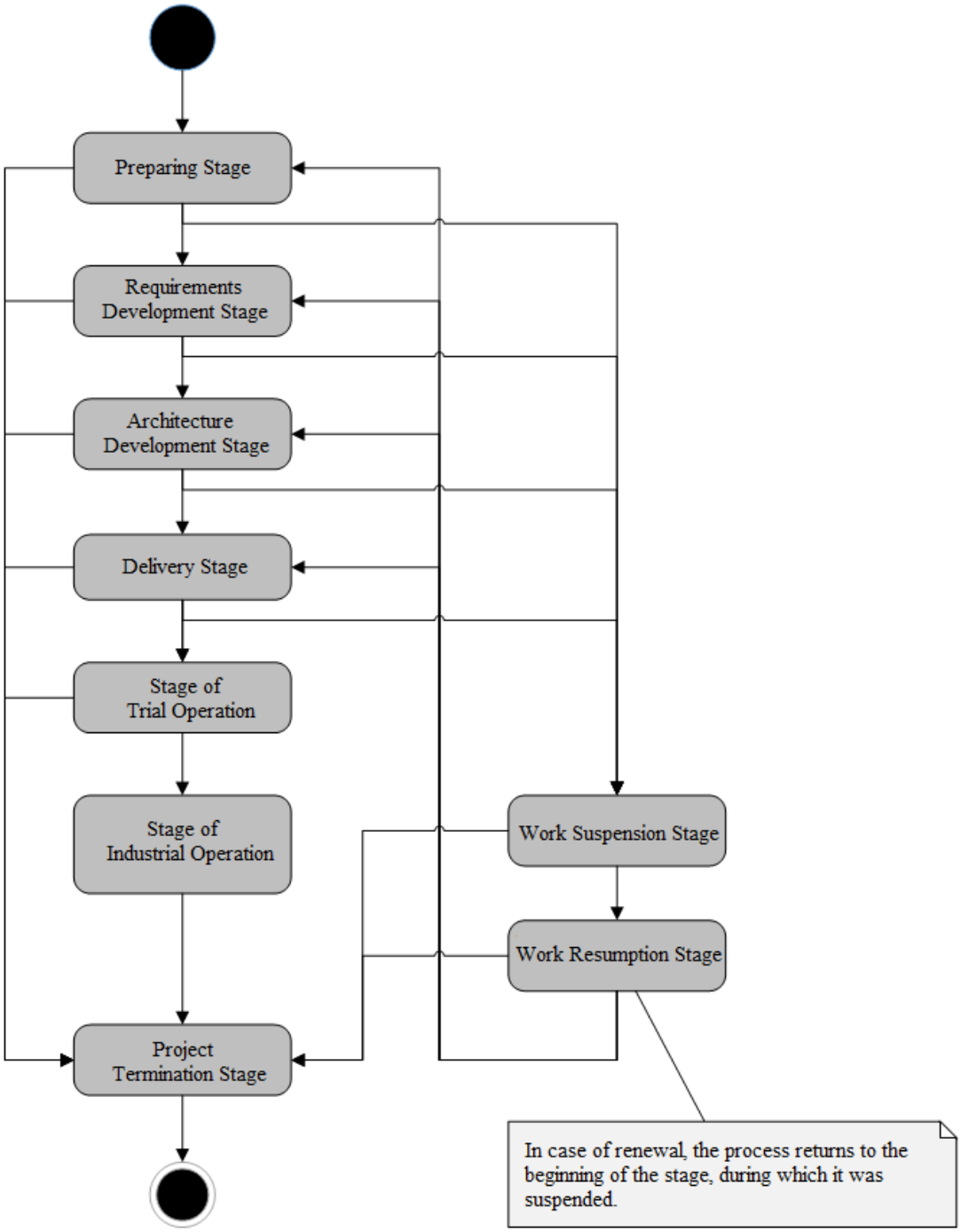
To begin, embedded software is different from customized software in that it is extremely difficult to maintain. Let's say you are releasing programs for refrigerators. After the software is delivered to the manufacturer, tens of thousands of devices begin to diverge around the world, and you have no idea where they will be. And if one of the refrigerators fails due to the fault of your software, it is easier to pay a penalty than to return the refrigerator to the factory and carry out diagnostics. Of course, you can prepare engineers for dealerships who can conduct on-site diagnostics and replace or update the software of a system, but this is still very expensive. Therefore, during developing that kind of software, several important limitations arise at once.

Firstly, the delivery is carried out in only one stage, for example, no one will build a half-working program into the devices.

Secondly, when delivering, developers also should focus on the quality of the program, since it will be very difficult to change it from the moment it is introduced into the iron box. Also, they should focus on the trial operation phase, when the program is implemented in a limited batch of devices, and these devices pass complex tests in various operating modes. They need to gather as much information as possible about the behavior of your system, analyze this information and refine the software.

Thirdly, the embedded software does not have a decommissioning phase. The program is simply thrown away with the device. Therefore, as soon as the batch of devices in which your software runs, the warranty period expires, you can proceed to close the project.

Finally, the embedded software development process is shown in the figure below.

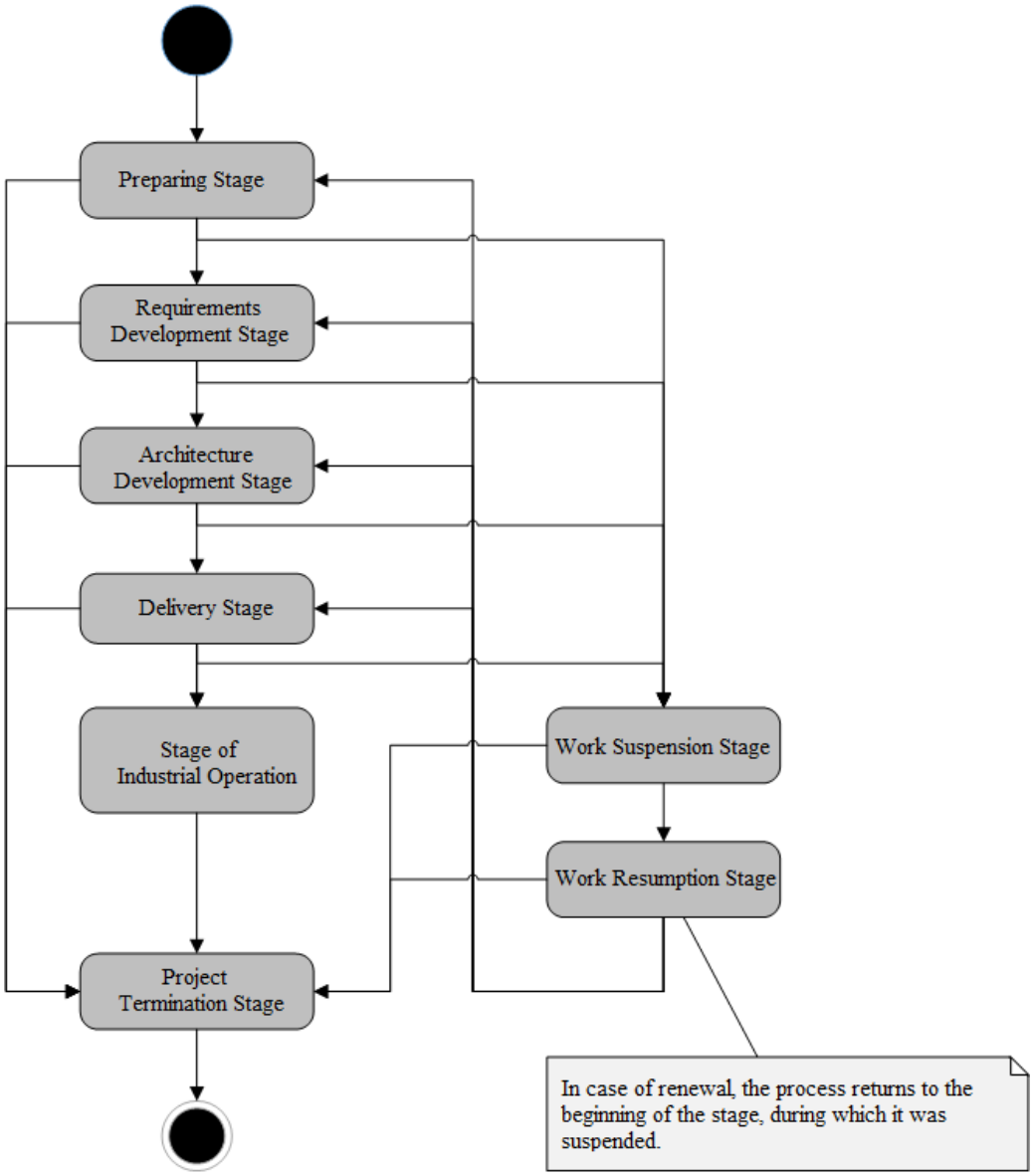


Games development process:

The game software was allocated by me because of the specifics of their production and operation. The gaming software business is based on the release of hits. One successful hit pays for the creation of several games that remain unnoticed by users. Therefore, the development process of one game is interconnected with the development processes of other games.

Another factor highlighting the production of games is the fact that the game is interesting to the user either until he passed the last level or until he had a fatal error. This means that he will not buy the second version of the game or even download it for free just to fix a few errors.

These factors affect the development of gaming software. The process is shown below.



It is necessary to note the following features of the process of developing gaming software.

First, the production of the games is very important to the quality of the concept. If the concept of the game does not allow you to create a hit, then further work is meaningless. The situation when most projects end in the preparatory stage is typical for developing gaming software.

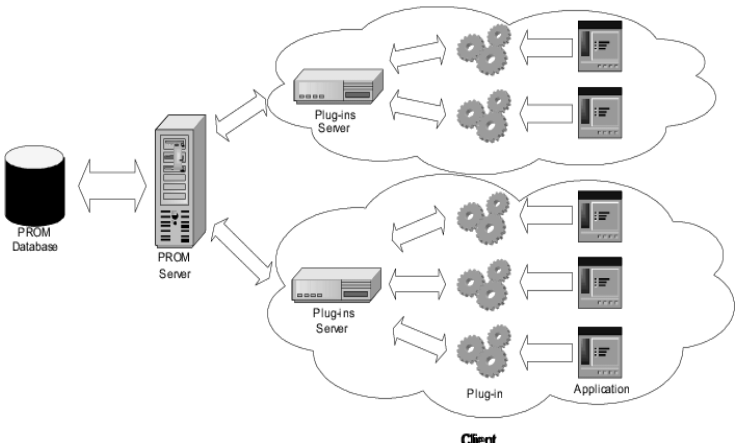
When developing requirements and architectures for gaming software, the lessons learned from previous projects are often reused. In this regard, the phase of the project also receives additional weight, when all useful developments should be recorded in the knowledge base of developers.

Metrics:

Since we have already become familiar with the types of software development processes, we will begin by selecting the appropriate computational platforms. After the company or startup decided to develop certain software, they also collected some necessary data, then it must choose a way to process them. Historically, indicators and analysis were mainly calculated using "PSP" and later using automated analytical programs deployed locally in the "PROM". Although firms can develop their own analytics system, although today many firms than ever been on the analytical data market.

Professional metrics:

1. **Personal Software Process**: It is a step-by-step application of the concept of development and quantitative management to the work of a developer in a learning environment. There are 4 main processes in “PSP”: “PSP#0”, “PSP#1”, “PSP#2” and “PSP#3”. Each process builds on the previous by adding engineering or managerial tasks. Step-by-step addition of tasks allows the developer to analyze the impact that new techniques have on his or her personal effectiveness. Usually given 10 tasks. PSP data is well suited for research, as many factors affecting project performance and introducing “interference” to research data, such as variability of requirements and teamwork difficulties, are either contained in the “PSP” or eliminated. The disadvantages of this analysis even exceed the imperfections of the qualitative analysis. Counting lines of code per hour is a very poor way to measure performance. However, it is impossible to say whether alternatives such as analysis of functional points, requirements or the number of user histories per hour are the best options, although all these four analyzes are used in software development projects. An alternative would be to analyze the number of hours spent on the task.
2. **Pro Metrics:** A more advanced method called "PROM" or "PRO Metrics" was described in the study "Collecting, Integrating and Analyzing Software Metrics and Personal Data Process Software at the end of 2003. "PRO Metrics" is a fully automated tool that is used in large companies to collect and analyze data obtained during process profiling software developers. This differs from the original model in that instead of the engineer collecting all the statistics on his own, data will be collected and analyzed without his participation. In short, this is an automated PSP that saves companies a lot of time and effort on analysis. However, "PRO Metrics" severely limits the types of metrics that are used for analysis. The program was installed on computers that collected data about the work done and sent it to the company's main server, where they were already analyzed by powerful processors and after the analysis was sent to a simple database that stores all developers and project data. The system is shown below.



3. **Analytics as a Service:** The method called "AaaS" is a universal use of Internet technologies for analyzing large amounts of data, using the advantages of online storage and using a third-party server for storing and analyzing data. The concept of "AaaS" comes from the ideas of "Software as a Service" and "Platform as a Service". For many years, large corporations store large amounts of data, but nowadays with the development of the world and technology, information has become too much for experts in the field of static analysis. It is necessary to process all this huge information flow for its more effective use. It is in this area that "Analytics as a Service" helps facilitate analysis for

companies, since it is a very large spectrum of services, such as special software like "SaaS", "PaaS", "IaaS", which in turn are models that replace traditional local systems on a web basis. For example, companies with “AaaS” instead of developing a large internal repository and its management software, can search for providers that offer access to a remote analytics platform for a small fee.

4. **Codacy:** Codacy is a startup in Lisbon, Portugal, which offers what it calls the “automated code review platform”. Founded in 2014, “Codacy” says it has been used by hundreds of companies, including the well-known as common customers like “PayPal”, “Adobe”, “Qlik”, “Cancer Research UK” and “Deliveroo”. The software can be installed on-site or is available in the cloud and is used by developers to check the quality of the code and implement the code quality standards. Co-founder of Codacy Jaime Jorge said, “Code review has become an integral part of any development workflow, and developers now spend more than 20% of their time analyzing code to catch errors and ensure quality as soon as possible” and he also believes that engineering teams will be 6% more efficient. Or, to use more tangible terms, it might be equivalent to giving a software fortnight ahead of any plan. Codacy, should help developers optimize about 30 percent of the time it checks code. In addition, Codacy have many customers from small digital consulting stores to large multinational corporations and cover several industries and geographic regions.

Small metrics:

5. **Focus Factor:** It shows how much the task should have taken, ideally, and how much it turned out as a result. "Concentration" of the team on the project. Can companies pay money based on this criterion? Possibly, but if your managers are not software engineers then programmers will consciously overestimate estimates in time, minimizing their own risks. The consequence of this approach is that the terms are stretched, the customer is indignant. In addition, each planning meeting will turn into squabbles and disputes in 10 minutes.
6. **Velocity = Focus Factor * Evaluation of new tasks:** Allows predict how many tasks the team will be able to solve in the next stage, depending on how much it has completed in the previous one. The problems are the same as for the focus factor, plus one more is added. Often the manager (especially inexperienced), who feels that the team’s performance can be “measured”, begins to use this tool “in the other direction”. But Velocity cannot be an exact criterion, since shows how long the same task can take, performed by the same team under the same conditions. However, after completing the task, the team has already changed: it has gained experience in how to specifically solve this task. And the metric will not work again.
7. **Cycle time:** How quickly time passes from the moment when the idea to implement a feature on a project came to the moment it was made. I personally love this metric. One of the key, which is worth measuring and optimizing. But developers do not directly influence this factor. This is too high-level metric. If company start to pay the team a salary based on what their “Cycle Time” is, it means that company, as a manager, do not strive to solve the team’s problems and understand the processes, but simply transfers everything to the developer’s team.

My opinion:

Different metrics should be measured and thought “intense thinking” influence them in positive way. But do not transfer high-level metrics directly to developers and designers.

Algorithms:

Over the past decades, self-driving cars, speech recognition systems and effective search have been created using machine learning. Now it is one of the most rapidly developing and promising areas at the junction of computer science and statistics, which is actively used in artificial intelligence and data science. Machine learning methods are used in science, technology, medicine, retail, advertising, multimedia generation and other fields.

1. **Machine Learning:** The use of analytical algorithms on the data stream is now one of the most urgent tasks in the field of building analytical systems. A classic example is neural networks, which are now very popular. Among them are networks associated with the classification. A simple example of a classification task is the definition of what is drawn in the picture: there is image, and we want to understand what it is: a dog, a cat, or something else. To write it with the help of standard code is very difficult, because it is not clear how to do it. Therefore, mathematical models are used, which are commonly called machine learning. They are created on the fact that certain patterns are extracted from many examples, and then, using these patterns, one can make predictions with some accuracy on new examples that were not in the original data set.
2. **Computational Intelligence:** The concepts of artificial intelligence brought a lot of remarkable practical results in terms of automation of human activity in various fields, which gradually changes the face of our civilization. However, the main goal - the creation of truly intelligent machines has not yet been achieved. At the same time, few of the scientists who really doubt that such a strong AI can be created in one form or another. If any objections sound, they have a religious character, appealing to the presence of a non-material soul in a person. But even with such radical views on the non-material world, they write off only such complex conceptually phenomena as free will, creativity or feeling, without denying the possibility of endowing the car with almost indistinguishable behavior from a person. Far less unambiguous are the answers to the questions, when and how exactly can a strong AI be created and in which practical areas it will help people. Nowadays, more realistically to achieve goal of the of the heuristic algorithms, which are suitable at solving some problems faster and with higher speed, than an any traditional algorithm is able. The ideas of heuristic algorithms come from the mathematical optimization and the aim of those algorithm is to find the suitable and not best solution basing on the information you have. Also, these algorithms can be associated with machine learning ideas, to create the machines for making logical decisions based on experience and existed data.
3. **How it can connect to the measuring of Software Engineering?** At this point in time, the goal of many companies like “Codacy” or “Codebeat” is to use machine learning methods in the field of automatic refactoring of object-oriented code and related research. To achieve this goal, the following tasks were set, we need to study how machine learning is currently used in code analysis. Select algorithms and metrics based on which will be implemented restructuring. Implement an automatic restructuring algorithm. Test the algorithm with training examples and real software products.
   Using algorithms for working with neural networks in programs written in object-oriented languages, can be found common design patterns. To implement a specific design pattern is usually required interaction of several classes, so the recognition process can be divided into two stages:
   - For each class in the program code, determine which function in each of the patterns in question, this class may play.

- Fixing the most likely combinations of class roles, determine do they together form one of the considered patterns.

As attributes, based on which training is carried out, the following characteristics can be used:

- "NOF"- the number of fields in the class.
- "NSF"- the number of static fields.
- "NOM" - the number of methods.
- "NOAM" - the number of abstract methods.
- "NOPC" - the number of private constructors.

More complex metrics can also be used, for example "RFC" (Response for Class) or "Ca" (Afferent Couplings). Another important task in the analysis of the code of object-oriented software applications is errors prediction without testing. To archive that different machine algorithms can be applied, for example, "Alternating Decision Tree" and "LogitBoost", which showed the best results compared with "Artificial Neural Networks", "Random Forest", "Naive Bayes" and "K-Star". In addition, automatic restructuring of programs can be carried out at the level of methods, classes or packages. In the first case of change occur in the structure of the method operators, in the second - the attributes and class methods must be reorganized between classes. When packet level restructuring some classes may be moved to other packages. Finally, those approaches which already exist in the restructuring of code can be applied for different languages and can be used in the creation of special indicator tool for evaluating the writing of software engineers' code.

Ethics:

Metrics and Monitoring:

Firstly, none of the currently known measurement models can boast of sufficient significance and accuracy. Metrics do not provide an objective picture of what is happening, they only provide some indicators calculated by a given algorithm. This is especially true in those cases where only one metric is entered to estimate a parameter. Only the use of a whole complex of metrics can give a complete picture. Therefore, metrics can unfairly measure individuals in some cases.

Secondly, often, if employees are aware of the indicators being measured, they try not to do their job properly, but rather adjust their indicators to the given parameters. For example, if "LOC" (the number of lines of code) is an important indicator, then programmers will write "Hindu" code, avoiding ways to simplify it, reducing the number of lines. Another example: if the indicator of the effectiveness of the work of programmers and testers is the number of admitted and identified bugs, then very soon they will agree among themselves how to "optimize" this indicator so that it is beneficial to both parties. Well, another example: if the project openly uses the focus factor metrics and if the project manager is not a "techie", then programmers are likely to specifically overestimate the time estimates to create a kind of "time security pillow" for themselves.

Thirdly, the usage and combining of metrics raise serious ethical issues. Such as, if the project simultaneously introduces metrics for individual employee assessment and metrics for evaluating the activities of the team, this can lead to internal conflicts. However, during optimizing their own performance, employees can "pull the blanket over themselves" and blame other team members for their failures. As a result, contradictions arise between the personal goals of the

employees and the overall goals of the team, the team does not work, and, as a result, the project is “inundated”.

Fourthly, the process of collecting metrics itself is fraught with certain technical difficulties. As a rule, for the collection of metrics and their interpretation are allocated, special staff. If we approach these processes without proper organization (especially as concerns the automation of metrics collection), then there is a risk that employees will simply “drown” in the collected information. Then you must attract additional people, and this is a direct path to excessive bureaucratization and waste of resources. In addition, there is a risk that due to the collection of numerical indicators, employees will be distracted from their immediate responsibilities. Of course, there are special automated systems for collecting metrics that simplify this process, but one way or another, you will still have to involve a human specialist who will have to spend some time on this work with metrics. Moreover, the time of the team members involved in the project is also spent on collecting metrics. And not every customer agrees to pay this time.

However, with proper implementation and proper interpretation, metrics can be a very useful tool for project managers. Also, using metrics which are tracking down the moments of a decline in the quality of development and determining causes and identify the most complex areas in the system. In addition, numerical indicators can help in planning the cost of resources for the further development of the product being developed.

Data Ownership and Monitoring:

A very important ethnic problem is storing and monitoring employee data in companies, which means that companies have direct access to emails, browser histories, screen content, even by pressing buttons on the keyboard. And they can use this data not only for commercial purposes, but also use this data to invade the personal life of employees. Nowadays, many developers are not aware of the rules for storing and using their information on a work computer and can use a work computer for their personal life. In my opinion, monitoring activity in companies should be transparent, adequate, and, above all, fair for all employees. The main conclusion is: it is tolerable to monitor what the employee does at the workplace, but with the observance of guarantees. It is necessary to warn him that he will be controlled, indicating what will be directly viewed, why (control objectives), to what extent and for how long. The results of the control cannot be used for other purposes, except for those that were reported to the employee. The main thing is that the control over the employee was carried out strictly with the above reasons. But at the same time, it is necessary to determine whether it is necessary to read his correspondence, mail, to catch his personal affairs at the workplace. If you can prove it without reading the personal correspondence, then the correspondence cannot be read.

Conclusion:

Within the framework of this report, I tried to make an overview of the “top level” of the process and measuring of developing application software. Each stage of the process, of course, needs a separate discussion with the obligatory consideration of the features of the software being developed. I note that the process diagram considered here is the result of summarizing my personal experience in developing various software tools. Like any generalization, my scheme is an abstraction. And, like any abstraction, it has its limits of applicability. You cannot mindlessly apply this scheme to a specific project. It is important to understand that each project has its own nuances that affect the organization of the development process. And therefore, for each project, the scheme presented here needs to be adapted, and in some cases, it will be necessary to develop a fundamentally different approach.