IMPROVING THE EFFICIENCY AND CONTROLLABILITY OF THE GLOBAL DEVELOPMENT OF GRAPHICAL INTERFACES ON WPF

Vladyslav Pashystyi

Abstract. *The issue of the correct distribution of work among team members has been an important part of software development for many years. Often the work must be divided even within one task. However, this can even degrade the speed and efficiency of execution. It is also likely to make the code unstable. This is especially true of the development of interface components.*

The solution lies on the surface - this is a good old organization of the process. Interested developers with affordable and clear development conditions are capable of fantastic efficiency.

Keywords*: parallel execution, command, requirements, tasks, components, interface, communication.*

DOI*: https://doi.org/10.54521/ijita28-01-p04*

Introduction

It is not always possible to do the right thing right when developing a software product. It is necessary to make a large number of decisions that can have consequences in both the short and long term. Sometimes these consequences result in a large loss of capacity of the development team and, as a consequence, a waste of money for the company and the customer.

Now almost all projects already use some tool to organize the interaction of team members and reporting. It may seem that this is a guarantee of controllability of tasks and their correct distribution among team members, but in reality, it is a much more complex process.

To begin with, it should be noted that the use of modern means of organizing interaction does not end with the creation of tasks and their distribution. As a rule, these systems have a lot of features, without the use of which, you can rarely effectively manage the project. Such tools include Jira, Codegiant, ClickUp, Asana, Monday, Wrike and many other alternatives. In the future, Jira will be used for examples.

Also, the type of task has a strong influence. Of course, there are more complex and easier tasks to organize the process. So, let's look at a complex case where developers need to work not only side by side, but almost on one piece of code. A good reflection of this situation is the development of interface components, on the example of the system for creating WPF interfaces.

Introduction to WPF interface components

We are accustomed to the fact that when developing applications, certain problematic issues, as a rule, someone has already found a solution. Thus, the implementation of WPF interface components has acquired several techniques depending on the expected performance of the component and its content.

Custom component - its essence is to imitate some basic class of the WPF system to give it new features, or edit existing behavior. Usually, the inherited component is already a ready-made component of the interface. Based on the description, you can already understand the purpose for which it is used - a small expansion of the basic components with easy further use. Running to the front, this type of problem solving is the most difficult to distribute among several team members.

User control is a combination of custom and basic components under one roof with a single management interface using Dependency Property. In terms of the complexity of the division of labor, it ranks second.

Few projects on WPF do without the use of the **MVVM pattern**. This application development technique separates the data to be displayed and their external representation. The resulting components will have multiple layers and usually include a large number of components and can even define an entire window, or include other components of any type. Of course, this approach is rightfully

the easiest for several team members to work simultaneously within one component..

Defining component requirements

It is extremely difficult to find any information about working closely on a particular area of code, but there is enough information about processing requirements. Of course, the writing of each component begins with the processing requirements, so we'll start with that.

Processing the extension of the requirements of an already started project, in the general case, can be divided into two radically different approaches. In the first case, the project manager, having received new requirements from the customer, can transfer responsibility for processing and execution on the shoulders of the team. In this case, the appropriate Jira tasks will be created, which include customer requirements, and the process of their distribution and implementation is entirely on the team. At first glance, this seems wild, but looking at the practice, there are projects that use this approach, have really good performance. This can only work thanks to a good team, that is, if there are responsible, diligent people who ideologically develop the project. It does not sound very plausible, but it really happens and not so rare, to be honest.

The second is much more difficult for the project manager. The method can even be divided into many stages. Among these stages:

1) new requirements are discussed in order to determine the timing, boundaries of responsibilities, problem areas, risk assessments and proposed solutions. All developers who are competent in the issues can participate in the discussion data.
2) each requirement is fixed in Jira. Preferably as a separate task. The results of the discussions are also recorded in the description or comments;
3) for each requirement the content and limits of realization are specified;
4) information analysis of the object of automation can be carried out;
5) a prototype design solution is formed (without implementation) to determine the components that will need to be developed or modified;

6) the results are agreed with the customer and in case of success, we move to the implementation. Tasks are reduced to a sufficient form to begin implementation. This includes setting priorities, executive programmers, testers, ordering documentation, and more.

The time required for implementation will be rationally discussed with the executing developers. In this case, the communication of developers with project managers or analyzers improves. This makes it much easier to give a truly accurate forecast of the required person / hours.

The task is desirable not to exceed 16 working hours. In case of violation of this limit, of course, it is better to create a separate task, or at least a subtask (supported in Jira). The reason for this lies in the error in determining the deadlines. For long-term tasks from 16 hours, setting the correct execution time is too complicated and often erroneous.

Note that only a person close to the project code can really assess the resolution of a component to be written by two developers. Thus, the team leader, project manager, analyst and other project members are unlikely to be able to properly share tasks without discussing with the developers themselves.

Separation of work on components

When the project already has some interface, then most often task is developing of the new functionality or extending the old one involves one component. However, keep in mind that this component can be an entire window model.

Such a task is sufficient to estimate the duration of 12 hours, so that it can already be considered as potentially suitable for several developers. Here the general problems of such cooperative work take over:

- communication problems;
- lack of experience for segregation of duties (work performed);
- great difficulty of division of responsibilities;
- the dependence of the process of one developer on the readiness to implement another;

– possible errors in the combination of work performed.

Communication, of course, is an extremely important part of this division of labor and must be up to par. Developers will need to talk about all expectations from each other. Today, almost all work has shifted to a remote level, so there are means of communication. This can be both synchronous and asynchronous communication. Asynchronous communication is when you send a message and do not expect to receive a response immediately. In contrast, synchronous communication is, for example, a live meeting or online conference. Of course, everything depends on the terms, conditions of implementation, the level of uncertainty of the requirements and many other factors. There is no universal method and it will most likely be necessary to use both.

You can even differentiate between implementation roles by simply specifying them in the Jira task itself. The main thing is to say the following in as much detail as possible:

– limits of work of each developer;
– procedure and terms of execution of parts;
– the need for additional communications at certain stages of readiness;
– assembling the finished parts of the component.

Developers may not even be able to do the work in parallel. For this reason, it is important to distinguish where parallel work may or may not take place. That is to determine the order of execution.

In this case, it is important to have at least one developer with good experience in working with the selected application support system and a good understanding of WPF itself. In its absence, the probability of making a mistake increases significantly, which can take a long time to correct and may require the intervention of more experienced developers.

Someone alone must take the initiative and become responsible. This is especially important when more than two developers are involved. His role as a developer will additionally include preparing for collaboration and assembling all parts into one working component.

Perhaps the biggest problem may be indivisibility of the task on several team members. This is mainly true for custom components, in which the code is mainly generated within one class. As a result, each developer must wait for another developer to complete work if they are running. Otherwise, the biggest problem will be the loss of one of the developers in the case of a very difficult involvement of changes to the predecessor. Although you might think that somehow you can get involved, but it's still a waste of time.

Even when all the pieces of the puzzle seem to come together, it is still necessary to check that the behavior of the parts is really predictable and meets expectations. Remember the golden rule that the sooner a bug is found, the better.

Team cohesion as the key to success

Interpersonal relationships in a team play a significant role in success. They definitely need to be improved during software development. Time management skills, communication skills, general teamwork skills, flexibility, emotional intelligence and independence will definitely help.

Time management is a science not only about time and punctuality. Time management is the ability to set priorities, calculate forces, break big tasks into smaller ones, plan step-by-step actions. It is important to learn to structure any of your tasks and plan resources for their implementation.

Communicativeness includes negotiation skills, presentations, the ability to find and discuss tasks and seek compromises. Communication skills will come in handy when communicating with colleagues, working together, sharing experiences and knowledge. If you want to develop, you can't go without it.

The ability to work in a team is another important soft-skill from the list of successful IT specialists. Of course, employees sitting in the office feel the team spirit much sharper than working remotely, but without teamwork in the IT field today, no way. Coordination and mutual understanding must always be established between the participants in the process, which is impossible to achieve without the experience of teamwork.

And even when you work remotely and your work is coordinated by a project manager, you still work in a team. Two people are already a team, and you must be able to cooperate and at least not let each other down.

When working on an IT product, significant adjustments are often made that need to be responded to quickly. The developer also needs to have a non-standard mindset to generate suggestions for product improvement.

Emotional intelligence has two equally important components: the ability to control one's emotions and the ability to understand other people's emotions. Emotional intelligence helps to quickly recognize the dissatisfaction or doubts of the interlocutor, the desires and expectations of others and instantly adapt their actions and emotions to the circumstances. Timely recognized dissatisfaction will allow to take measures and prevent the development of conflict within the team.

Last but not least is understanding the importance of the skill of doing more than you are expected to do and not asking questions that you can answer yourself.

Preparation of tasks and code before implementation

We have already determined that it is necessary to have the initiative of one of the developers. This also means that this team member not only monitors the final result, but also prepares everything to start working together.

Preparation takes place in three stages: discussion, indicating the results of the discussion and writing the source code.

During the discussion we determine: the boundaries of each developer, the order and timing of parts, the need for additional communications at certain stages of readiness and integration of the finished parts of the component.

All developers who will be directly involved in the implementation and, if necessary, other competent team members or the project manager must participate. For the most part, this depends on the usual routine of the team.

A separate subtask is created for each of the performers in which there will be a short and clear description of his role. We also add general provisions to the description of the main task. The wording should be clear to all participants.

Writing source code has only one critical goal - minimizing developer code intersections. It is understood that each developer can edit a particular block of code with some confidence that it will not interfere with other developers when they combine this block of code with their progress.

This must include a basic set of fields, methods, classes, and interfaces, depending on the type of component.

The **custom component** by its nature is very complex to divide the process of its implementation into several simultaneous performers. This is due to the fact that often changes are made almost entirely within one class, so developers can not help but interfere with each other.

Suppose you still need to divide the work, because it is urgent to make a certain component.

In this case, it is good when you cannot do it in one class. For example, when a certain functionality is logically allocated to a separate class. You can create mock methods without implementation for selected classes and this will be enough to start parallel work.

Otherwise, the most problematic is the division of labor within one class. This rarely goes smoothly and ideally should never be done. You can also create methods in advance and start filling them in parallel. When the code is combined, a conflict will occur. Hope that the conflict resolution will be automatic, because otherwise you will have to manually sort the whole class in the conflict resolution tool.

The **user controls** is more distributed due to the use of combinations of components in the middle. This adds the ability to divide the work by nested components. It may be that the components are too dependent on each other and here, unfortunately, the principles begin to work as for custom components. However, unlike custom components, here the best divisibility can be not only

due to the use of a set of other components in the middle, but also in general by classes and interfaces.

MVVM provides maximum freedom of work distribution. Reducing the connection within each model even allows you to conveniently divide the work into creating a display and organizing the data for that display. To do this, you only need to set the specific names of the properties of the model with the data.

However, the separation of mapping and mapping model does not end well. Along with MVVM comes the ability to more conveniently use many other patterns such as GoF, SOLID, DI and many others. All of them can help to better find the boundaries of each developer.

We can pay special attention to the last principle of SOLID and a special case - DI. Reducing connectivity and dependency mostly leads to the use of interfaces, which is a kind of clear allocation of methods needed to describe a particular developer. Similarly, all this helps a lot in the subsequent final integration of the code, as the code is in different classes that only implement interfaces.

The greater the argumentation, clarity and elaboration of the requirements, the better the parallel work will be. The maximum level is complete parallelism after preliminary training.

If an unprocessed moment is found in the requirements, it may be a misunderstanding of what specific methods, properties and interfaces are needed. As a result, at one point in time, parallel streams will need to be re-synchronized for retraining. There may be many such synchronizations, depending on the inaccuracies found in the initial requirements.

Results

It is logical that the effectiveness strongly depends on the certainty of the initial requirements, the level of communication with the customer and other similar factors. However, if you follow the above procedures and recommendations, then, under equal conditions, the increase in the speed of work will be up to one and a half times.

Speed, of course, is a good indicator, but it will also improve quality, which is no less important. There will be a significant reduction in the number of errors made and not found in the implementation. This indicator can even be considered more important, because the mistake causes problems for the customer, spends money and time to correct.

To all this is added the increase of controllability of parallel work, which in any case will be well reflected in the level of maturity of the project development process.

Conclusion

Therefore, the organization of the workflow is an important part of the simultaneous work of developers on a certain part of the code. Starting with clarifying and communicating the initial requirements to developers, ending with checking the finished code component.

It is necessary:

- Improving the transfer of requirements to developers, including communication with the developers themselves and good design of tasks in modern systems of interaction (for example, Jira);
- good interpersonal communication of developers;
- clear definition of the boundaries of each developer;
- determination of the responsible person;
- minimizing the dependence of developers on each other by clarifying the requirements;
- improving interpersonal relationships in the team;
- significant preparation of code and tasks before the implementation.
- verification by the responsible person of the fulfillment of the set requirements after the combination of the component.

By organizing the process in detail, you can greatly increase the controllability of development in difficult conditions of joint code writing. As a result, we obtained results that increase the speed, quality and controllability of the tasks.

Bibliography

[Nathan, 2011] Adam Nathan WPF 4 Detailed Guide. Symbol-Plus, 2011. ISBN — 978-5-93286-196-7.

[MacDonald, 2013] [electronic resource] MacDonald Matthew. WPF: Windows Presentation Foundation in .NET 4.5 with examples in C # 5.0 for professionals. Williams, 2013. ISBN 978-5-8459-1854-3.

[Habr, 2019] Habr. JIRA as a remedy for insomnia and nervous breakdowns. 2006. https://habr.com/ru/company/lanit/blog/464497/

[Weil, 2016] Weil Arnaud. XAML, C# and the MVVM pattern. Learn WPF MVVM. Leanpub, 2016, 165 p

[Andrade, 2007] Andrade Chris, Livermore Shawn, Meyers Mike, Van Vliet Scott. .NET Development with the Windows Presentation Foundation. Professional WPF Programming. Wiley Publishing Inc., Indianapolis, Indiana, 2007, 452 p.

Authors' Information

Vladyslav Pashystyi – Senior .Net Core, C#, WPF Developer

Bachelor. Faculty of Software Engineering, Department of Software Engineering, National Aviation University, Kiev, Ukraine.

Kyiv, Ukraine.

Major Fields of Scientific Research: Architectural programming, Responsive user interfaces.

E-mail: pvladvoland@gmail.com