M. D. VERES, Student, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine; e-mail: maksym.veres@nure.ua; ORCID: https://orcid.org/0009-0009-1768-8693

N. V. GOLIAN, Associate Professor of the Department of Software Engineering, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine; e-mail: nataliia.golian@nure.ua; ORCID: https://orcid.org/0000-0002-1390-3116

OPTIMIZATION OF THE DEVELOPMENT PROCESS OF MONOLITHIC MULTI-MODULE PROJECTS IN JAVA

In recent years, there has been an increase in the complexity of Java software development and a change in the scope of projects, including an increase in the number of modules in projects. The multi-modularity of projects, although it improves manageability to a certain extent, but often creates a number of problems that can complicate development and, a problem that will appear in the future, require more resources to support. This article will analyze the main problems of monolithic multi-module Java projects and will try to consider a number of possible solutions to overcome the above problems. The article discusses the peculiarities of working with multi-module monolithic projects using Java as the main programming language. The purpose of this article is to identify features and obstacles using the above architectural approach of the software, analysis of the main possible issues of working with the monolithic multi-module Java projects, as well as providing recommendations for eliminating these obstacles or describing the features of the process that could help engineers in supporting this kind of projects. In other word the main goal of this work is to create recommendations, provide modern best practices for working with monolithic multi-modular software architecture and the most popular modern technological solutions used in corporate development. The proposed recommendations allow the team, primarily developers and the engineering side, to avoid possible obstacles that lead to the loss of efficiency of the monolithic software development process. The most important advantage, from the recommendations given in the article, is the optimization of resource costs (time, money and labor) for the development process. As a result of the article, a general list of recommendations was obtained, which allows the developer to better analyze what changes in the project should (if necessary) be made to optimize the development, assembly and deployment processes of a monolithic Java project, as well as advice before designing new software to avoid the main obstacles of monolithic architecture in the future.

Keywords: monolithic architecture, multi-module architecture, Java, project build, module, development, project deployment.

Introduction. To begin with, here is a short description of monolithic architecture and its features. To do this, let's turn to the article «Microservices vs. monolithic architecture» by Atlassian marketing strategist Chandler Harris [1].

A monolithic application is a single common project, while a microservice architecture, in turn, is a combination of small, independently deployed services. Monolithic architecture has already become a traditional software model; it represents a single project that works autonomously and independently of other applications. In turn, the code base in such an architecture combines all business tasks.

Let us highlight the main advantages and disadvantages of monolithic architecture. The advantage is the ease of deploying applications with a monolithic architecture. Fewer independent modules reduce configuration and deployment costs. Using a single code base in some situations leads to simplified development, however, making fundamental changes to its structure can, on the contrary, significantly increase costs.

In some situations, the performance of monolithic applications can be higher than that of applications with a microservice architecture. For example, in a centralized codebase it is sometimes possible to use a single API, which often immediately sends a request to the data storage system, when, in turn, a microservice architecture requires calling various APIs, as well as transferring data through application communication interfaces (for example, REST).

In cases where at the initial stages, as the monolithic application grows, its main disadvantages begin to appear.

One of the main ones is a significant increase in resource costs for development (both material and time). Especially, this flaw will manifest itself during major changes in a single code base, which may affect all functionality. Not only does the developer need to take this into account, but also a competent team lead must realize that it is necessary to almost completely test the functionality of the project, which could be affected by these changes. This is a rather unpleasant factor, especially during the period when, for example, the release date is approaching, and changes to the code base are requested and occur after the main business functionality has been tested. Thus, there is a need to retest the application, where there is a risk that it will not be fully completed by the release date.

As a possible example, a situation where the business side urgently requires new functionality, in addition to what was planned in advance, and from the technical management side, closer to the end of the development period, a request comes for a mandatory update of the project build system plugins to close a technical vulnerability. After updating the plugins, part of the project's functionality stops working with an error. The development team needs to try to fix the problem with urgency, and the testing team has even more urgency to verify that the problem is fixed.

The next disadvantage of such an architecture - unreliability. An instability or bug in one module has the risk of affecting the availability of the entire application.

Scalability is the third key problem. Individual components cannot be scaled, since they are part of a single code base and are not presented, at least, as a separate stand-alone module [2].

80

Вісник Національного технічного університету «ХПІ». Серія: Системний аналіз, управління та інформаційні технології, № 1 (11) '2024

About multimodularity. Let us briefly recall what multimodularity is, why this approach is used, and what advantages and disadvantages it has. To do this, let's turn to Google's article on using the modularization approach in Android development [3].

And so, modularization is a practical approach that consists of breaking the project's code base into loosely coupled and autonomous parts, each of which is a separate module. Each such module is independent and created for a clear purpose. For example, a simple application that has a presentation module, a module that represents business logic, and a module for communicating with the data storage environment. Thus, dividing the code base into modules can reduce the complexity of designing and maintaining especially large systems. Let us highlight the main advantages of this approach.

The multimodular approach opens up the possibility that a module's code base can be reused both within a project and across multiple applications. In this case, modules are individual building blocks. An example would be where an application has multiple presentation modules, say a UI module and multiple REST API modules. In this case, they can reuse the functionality of the module (modules) for working with business logic, which in turn uses the module for working with the data storage environment.

Another benefit is increased efficiency in dependency management. It becomes possible to control which module dependencies and which part of its code base needs to be transferred to other modules, and which needs to be hidden or encapsulated.

The next advantage is scalability. When using a properly implemented principle of separation of concerns, changes in one module will not cause a cascade of changes in all other modules of the project [4]. However, often, in practice, in both old and new monolithic projects, this principle is not taken into account, which subsequently significantly increases support resources.

However, in the process of implementing a modularization approach, developers can often encounter risks and common architectural mistakes. For example, each module will introduce additional costs in the form of increased assembly complexity and cumbersome boilerplate code. At the same time, the complexity of the module assembly configuration creates the risk of making it difficult to maintain configuration consistency across all project modules in the long term. The costs of implementing this approach should be assessed to see if they will hinder the improvement in scalability. In such cases, a possible solution would be to conduct an architectural refactoring to find modules that could be consolidated.

Conversely, in some cases, especially on large monolithic multimodule projects that are already quite old, the problem arises that the modules can become too large, thereby spawning another monolith, which in turn deprives almost all the benefits of the multimodularity of the project. For example, business logic modules quite often grow in applications, but some functionality is used only once, at the same time requiring quite a lot of configurations and resource costs [5].

The next danger in implementing a multimodule approach is unnecessary complexity. It is necessary to analyze whether, in principle, it makes sense to split the project into modules. The main thing to pay attention to is the size of the codebase. If the project is not expected to exceed a certain size limit, the scalability gain will be either so small or non-existent that implementing the approach discussed will make no sense. As an example, there is no point in splitting very small microservices that perform one single simple action into modules.

To summarize, to determine whether multimodularity is suitable for a particular project, you can only first assess its expected size, the need for scalability, encapsulation, or simply reduce build time, then it makes sense to consider the possibility of switching to a multimodular architecture.

Multimodularity in Java. Java is one of the most popular object-oriented programming languages today [6]. Generally, more commonly used today for either enterprise development or Android development. In addition to a number of advantages that this language has, one of them has only recently begun to appear in a real development environment - the Java Platform Module System (JPMS). Let's look at how this works, using an article from the Baeldung resource [7] as well as material presented in the book «The Java Module System» [8].

JPMS arrived with the 9th release of Java and brought with it a new layer of abstraction over packages known as the Java Platform Module System.

A module in JPMS itself is a group of closely related packages and resources, along with a newly introduced module descriptor file. The packages within each such module are identical to those already existing in Java. Each module is responsible for its own resources, such as, for example, media or properties and configuration files.

At the time of writing, there are four types of modules: system modules - modules that are listed when the list-modules command is run. They include Java SE and JDK modules. Application modules are those modules that are usually created when there is a need to use multimodularity. They are defined in the compiled «Module-info.class» file included in the compiled JAR file. Automatic modules are unofficial modules that can be added to a project by appending existing JAR files to the module path. The module name will be derived from the name of the JAR file. Automatic modules receive full read access to all other modules that have been written to the path. Unnamed Module - In situations where a class or JAR file is loaded on the classpath but not on the module path, it automatically becomes part of the unnamed module. Thus, it is a universal module that allows backward compatibility with previously written Java code.

Modules have two distribution methods. The first is in the form of a JAR file or in the form of a «disassembled» compiled project. It is also possible to create multimodule projects consisting of a «main program» and several library modules. When creating new modules, you need to make sure that there is only one module in each JAR file. When setting up the build file, it is necessary to combine each project module into a separate JAR file. To configure the module, you need to place a specialized file named Module-info.java in the root of the packages. This file is

Вісник Національного технічного університету «ХПІ». Серія: Системний аналіз, управління та інформаційні технології, № 1 (11)'2024

81

known as a module descriptor, containing all the data necessary to create and use a new module.

It is worth highlighting a number of problems associated with using the Java Platform Module System. Large number of projects (especially in the corporate environment) are currently avoiding the implementation of this technology due to the difficult migration of existing projects, since it requires quite significant resources and changes. Also, managing dependencies between modules can add complexity to the development process. JPMS requires explicit definition of dependencies, which in turn creates the possibility that the project structure will become more granular and more additional code will need to be written to manage dependencies. Another problem is the limited support for JPMS by the currently extremely popular Maven and Gradle build and deployment systems, which often results in a refusal to implement JPMS into a project. Quite often third-party libraries and frameworks do not have support with JPMS, in cases where their dependency structure is quite complex, or the Java Reflection API is used. In this case, it is necessary to create additional «adapters» that will allow the implementation of JPMS. In some cases, the use of JPMS, like any complex module management structure, can lead to a drop in performance and additional resource costs, which can be especially sensitive for some projects.

Perhaps, the lack of sufficient experience with this technology and, accordingly, its support by third-party frameworks and code makes JPMS a rather controversial option for implementation as a project modularization system, inferior in capabilities to the same tools for assembling and deploying applications such as Gradle and Maven.

Popular solutions. Let's look at popular solutions to improve the process of working with monolithic multimodule applications. The very first and obvious thing is to use popular project build systems for Java projects, such as Apache Maven and Gradle.

Apache Maven is a dependency management and build automation tool primarily used for Java projects [9]. The core concepts of Maven include the POM (Project Object Model) approach, which is an XML file that describes the project structure, its dependencies, plugins and other settings. This file is the main configuration element for Maven and, in turn, contains project information such as name, version, dependencies, etc.

The next concept of Maven is the automatic downloading and management of dependencies, which are specified in the POM file and downloaded automatically using the central Maven repository or other remote repositories. This process build system contains a division of the project build process into several phases, which include compilation, testing, packaging, deployment and others. Each such phase is performed in a specific project and can be configured using plugins. Plugins, in turn, allow you to expand functionality by introducing new build goals or performing other tasks. Maven has both built-in plugins and support for creating your own plugins.

Let's look at how to use Maven for a new project. To do this, we will generate code using the application archetype «maven-archetype-quickstart». We will use the next example command:

mvn archetype:generate -DgroupId=com.example -DartifactId=my-project -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

Further, as the code is written, the necessary dependencies will be added to the project, for example, the JUnit testing framework. To do this, we need to edit the «pom.xml» file and declare the dependency in it. The example for JUnit dependency is displayed on fig. 1.

```
<dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
</dependency>
```

Fig. 1. The example of the «pom.xml» modification

And when everything is ready, we can run the following command to get the project JAR file in the local Maven repository «mvn clean install».

Gradle is also a tool for automating project build, testing, and deployment, with an emphasis on a declarative paradigm as opposed to an imperative one [10]. The project structure and its dependencies are described using DSL (Domain Specific Language). This tool is more flexible in describing the project structure; DSL can be used based on Groovy, Kotlin. There is also support for parallel builds, incremental builds, the ability to create separate tasks, and much more. For example, the ability to define various assembly tasks, configure dependencies, source code sources, resource management, etc. While Maven can also be used for monolithic, multimodule Java projects, Gradle, due to its flexibility and rich customization options, is often preferred for more complex projects with a large number of modules and components.

Let's look on at a simple example of how Gradle allows us to create a task to package, test, further build and deploy a JAR file of a specific module to a specific path. The example is on fig. 2.

The image describes the general example structure of the «build.gradle» files. With defining the plugins, group, version, module dependencies, configured repository sources and specific tasks created manually. This file is placed in each module of the project supported by Gradle build system. The configuration properties could be changed in the «gradle.properties» file placed in resources.

This «build.gradle» file defines the following tasks. «buildModule» – the task that builds the module will first execute the «clean» (to clean up previous builds) and «test» (to run tests) commands, after which a JAR file will be created using the corresponding «Jar» plugin. «testModule» is a task that runs unit tests for the module. «deployModule» – this task deploys the assembled JAR file to the specified directory. Thus, you can execute all three commands by running only one of them.

82

Вісник Національного технічного університету «ХПІ». Серія: Системний аналіз, управління та інформаційні технології, № 1 (11)'2024

```
plugins {
        id 'java'
}

group 'com.example'
version '1.0-SNAPSHOT'

repositories {
        mavenCentral()
}

dependencies {
        testImplementation 'junit:junit:4.12'
}

task buildModule(type: Jar) {
        dependsOn 'clean', 'test'
        archiveFileName = "${project.name}-${project.version}.jar"
        destinationDir = file("$buildDir/libs")
        from sourceSets.main.output
}

task testModule(type: Test) {
        testClassesDirs = sourceSets.test.output.classesDirs
        classpath = sourceSets.test.runtimeClasspath
}

task deployModule(type: Copy) {
        dependsOn 'buildModule'
        from "$buildDir/libs"
        into 'path/to/deployment/directory'
}
```

Fig. 2. The example of the «build.gradle» file

Gradle has incremental (enabled by default) and parallel project builds.

Parallel assembly, while reducing time costs, especially for multicore systems, can also significantly increase the required computing resources of the system where it occurs.

And the next recommendation is to use Docker to simplify the deployment process [11]. The use of this technology will bring a number of advantages to the project. The ability to flexibly configure Docker files for each module has a positive impact on the scalability of the project.

Perhaps one of the few disadvantages of this technology is the complexity of configuring the Docker images themselves, especially for beginners and in cases where the project has a complex structure with a large number of dependencies. Docker containers themselves will require additional computing resources to operate. This may result in increased costs for maintaining the application deployment environment [12].

Thus, Docker may introduce additional costs for maintenance and support, including container orchestrators (for example, Kubernetes). However, for large projects this is a good solution that will make the deployment process much more efficient.

Let's look at how to use Docker in practice. Let's imagine that we have a monolithic project in Java, which has several modules, let's call them abstractly «repository», «api», «service», «web». First of all, we will define a Dockerfile for each of the modules. The example of the Dockerfile for «api» is displayed on fig. 3.

The next step will be building and running the Docker images for each of the modules with the next type of commands. «docker build -t myproject-api:1.0 api/» and «docker run -d --name api-container myproject-api:1.0» And this is how we get a running container for the «api» module in the background.

```
# Using a base JDK image
FROM openjdk:11
# Copy a JAR-file from the project build to the container
COPY build/libs/api.jar /app/api.jar
# Define a command for project start
CMD ["java", "-jar", "/app/api.jar"]
```

Fig. 3. The example of the Dockerfile

Conclusions. The result of this article, is examined features of working with multimodule monolithic Java projects.

Listed the key problems of this type of project architecture, ways to solve them, as well as technologies and tools that can improve the efficiency of the development and deployment of these projects, and gave practical examples of their use. Discovered the ways to replace the monolithic architecture with the microservice architecture.

As a summary, the main recommendation discovered in this article is to get rid of most of the problems is to switch to a microservice architecture whenever possible, use the Gradle project build tool and Docker containerization. This makes the development process more effective and reduces the project support costs.

References

1. *Atlassian. Microservices vs. monolithic architecture.* URL: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith (дата звернення: 3.05.2024).
2. Martin R. *Clean architecture.* Prentice Hall, 2017. 432 p.
3. *Microservices.io. Pattern: Monolithic Architecture.* URL: https://microservices.io/patterns/monolithic.html (дата звернення: 10.05.2024).
4. Fowler M. *Patterns of Enterprise Application Architecture.* Addison-Wesley Professional, 2003. 560 p.
5. *Android Developers. Guide to android app modularization.* URL: https://developer.android.com/topic/modularization (дата звернення: 10.05.2024).
6. Bloch J. *Effective Java.* Addison-Wesley Professional, 2017. 416 p.
7. Franklin C. *A guide to java 9 modularity. Baeldung.* URL: https://www.baeldung.com/java-9-modularity (дата звернення: 10.05.2024).
8. Parlog N. *Java module system.* Manning Publications Co. LLC, 2019. 440 p.
9. Lalou J. *Apache Maven Dependency Management.* Packt Publishing, 2013. 158 c.
10. Muschko B. *Gradle in action.* Manning Publications Co. LLC, 2014. 480 p.
11. Mouat A. *Using docker: developing and deploying software with containers.* O'Reilly Media, Incorporated, 2015. 354 p.
12. Oggl B., Kofler M. *Docker: practical guide for developers and devops teams (the rheinwerk computing).* Rheinwerk Computing, 2023. 491 p.

References (transliterated)

1. *Atlassian. Microservices vs. monolithic architecture.* Available at: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith (accessed 10.05.2024).
2. Martin, R. C. *Clean Architecture.* Prentice Hall, 2017. 432 p.
3. Microservices.io. *Pattern: Monolithic Architecture.* Available at: https://microservices.io/patterns/monolithic.html (accessed 10.05.2024).
4. Fowler M. *Patterns of Enterprise Application Architecture.* Addison-Wesley Professional, 2003. 560 p.
5. *Android Developers. Guide to Android app modularization.* Available at: https://developer.android.com/topic/modularization (accessed 10.05.2024).
6. Bloch, J. *Effective Java.* Addison-Wesley Professional, 2017. 416 p.

Вісник Національного технічного університету «ХПІ». Серія: Системний аналіз, управління та інформаційні технології, № 1 (11)'2024

83

7. Franklin, C. *A Guide to Java 9 Modularity. Baeldung.* Available at: https://www.baeldung.com/java-9-modularity (accessed 10.05.2024).
8. Parlog, N. *The Java Module System.* Manning Publications Co. LLC, 2019. 440 p.
9. Lalou, J. *Apache Maven Dependency Management.* Packt Publishing, 2013. 158 p.
10. Muschko B. *Gradle in action.* Manning Publications Co. LLC, 2014. 480 p.
11. Mouat A. *Using docker: developing and deploying software with containers.* O'Reilly Media, Incorporated, 2015. 354 p.
12. Oggl B., Kofler M. *Docker: practical guide for developers and devops teams (the rheinwerk computing).* Rheinwerk Computing, 2023. 491 p.

УДК 004.41

М. Д. ВЕРЕС, студент, Харківський національний університет радіоелектроніки, м. Харків, Україна, 61166; e-mail: maksym.veres@nure.ua; ORCID: https://orcid.org/0009-0009-1768-8693

Н. В. ГОЛЯН доцент кафедри програмної інженерії, Харківський національний університет радіоелектроніки, м. Харків, Україна, 61166; e-mail: nataliia.golian@nure.ua; ORCID: https://orcid.org/0000-0002-1390-3116

ОПТИМІЗАЦІЯ ПРОЦЕСУ РОЗРОБКИ МОНОЛІТНИХ БАГАТОМОДУЛЬНИХ ПРОЄКТІВ НА JAVA

В останні роки спостерігається зростання складності розробки програмного забезпечення на Java та зміна обсягу проєктів, у тому числі збільшення кількості модулів у проєктах. Багатомодульність проєктів хоча й покращує певною мірою керованість, але часто створює низку проблем, які можуть ускладнити розробку та, проблему, яка з'явиться в майбутньому, вимагати більше ресурсів для підтримки. У цій статті буде проаналізовано основні проблеми монолітних багатомодульних Java-проєктів і зроблена спроба розглянути ряд можливих рішень для подолання вищезазначених проблем. У статті розглядаються особливості роботи з багатомодульними монолітними проєктами з використанням Java як основної мови програмування. Метою даної статті є виявлення особливостей і перешкод при використанні вищезазначеного архітектурного підходу програмного забезпечення, аналіз основних можливих проблем роботи з монолітними багатомодульними Java-проєктами, а також надання рекомендацій щодо усунення цих перешкод або опису особливостей процесу, який може допомогти інженерам у підтримці такого роду проєктів. Іншими словами, основною метою даної роботи є створення рекомендацій, надання сучасних кращих практик роботи з монолітною багатомодульною архітектурою програмного забезпечення та найпопулярнішими сучасними технологічними рішеннями, які використовуються в корпоративній розробці. Запропоновані рекомендації дозволяють команді, насамперед розробникам та інженерній стороні, уникнути можливих перешкод, які призводять до втрати ефективності процесу розробки монолітного програмного забезпечення. Найважливішою перевагою, з рекомендацій, наведених у статті, є оптимізація витрат ресурсів (часових, грошових і трудових) на процес розробки. У результаті статті отримано загальний список рекомендацій, який дозволяє розробнику краще проаналізувати, які зміни в проєкті необхідно (якщо необхідно) внести для оптимізації процесів розробки, збірки та розгортання монолітного Java-проєкту, а також поради для розробки нового програмного забезпечення, щоб уникнути основних перешкод монолітної архітектури в майбутньому.

Ключові слова: монолітна архітектура, багатомодульна архітектура, Java, збірка проєкту, модуль, розробка, розгортання проєкту.

Повні імена авторів / Author's full names

Автор 1 / Author 1: Верес Максим Дмитрович / Veres Maksym Dmytrovych
Автор 2 / Author 2: Голян Наталія Вікторівна / Golian Natalia Viktorivna

84

Вісник Національного технічного університету «ХПІ». Серія: Системний аналіз, управління та інформаційні технології, № 1 (11)'2024