«Сейфуллин оқулары-18(2): «XXI ғасыр ғылымы – трансформация дәуірі» Халықаралық ғылыми-практикалық конференция материалдары = Материалы международной научно-практической конференции «Сейфуллинские чтения – 18(2): «Наука XXI века - эпоха трансформации» - 2022.- Т.I, Ч.III. - Р.160-164.

MDA DEVELOPMENT PROCESS. PROS AND CONS

Karpenko N., student

University of information technology and Management, Rzeszow, Poland

Toleuova S., Senior Lecturer

Seifullin Kazakh AgroTechnical Univercity, Nur-Sultan

In terms of maturity, hardware development is often compared to software development. It can be noted that, in the development of hardware, significant progress has been made, for example, the processor speed has grown exponentially in twenty years, and the progress made in software development seems minimal. To some extent this is a matter of phenomena. The progress made in software development cannot be measured in terms of development speed or costs. The growth in software development is evident from the fact that it is possible to build much more complex and larger systems. Just imagine how quickly and efficiently we can build a monolithic mainframe application that does not have a graphical user interface and will not be connected to other systems.

We never did this again, so we do not have solid figures to support the idea that progress has been made. Nevertheless, software development is an area in which we are struggling with a number of serious problems. Writing software is time-consuming. With each new technology you need to work hard again, and again. Systems are never created using only one technology, and systems must always interact with other systems. In other words, there is a problem of constantly changing requirements.

The software development process as we know it today is often driven by low-level design and coding. A typical process, as illustrated in Figure 1 includes number of phases [1]:

1. Conceptualization and requirement gathering
2. Analysis and functional description
3. Design
4. Coding
5. Testing
6. Deployment

Whether we use an incremental and iterative version of this process or a traditional waterfall process, documents and diagrams are created during stages 1

through 3. They include a description of the requirements in the text and images, and often many Unified Modeling Language (UML) diagrams, such how use cases, class diagrams, interaction diagrams, activity diagrams, etc. The paper stack is sometimes impressive. Nevertheless, most of the artifacts from these phases are just paper and nothing more. As soon as coding begins, all documents and corresponding diagrams created in the first three phases quickly lose their values. The union between the diagrams and the code disappears as the coding phase progresses. Instead of being an exact specification of the code, the charts usually become more or less unrelated images. Each change in the system entails changing the distance between the code and the text and the diagrams created in the first three phases. Changes are often performed only at the code level, because the time to update charts and other high-level documents is not available. In addition, the added value of updated charts and documents is questionable, because all new changes begin with the code in any case. Extreme programming has become a popular quick way. One of the reasons for this is that he believed that code is the most important part of software development. Because the only phases that were really productive are coding and testing. This means that the XP approach solves only part of the problem. While one and the same team is working on software, understanding the system is enough knowledge in their heads. During initial development, this is often the case. The problem begins when the command is dismounted, which usually occurs after the delivery of the first version of the software. Other people should support (fix bugs, improve functionality, etc.) software. Having only code and tests, it is very difficult to maintain the software system. So, either we use our time in the early stages of software development, creating high-level documentation and diagrams, or we use our time in the maintenance phase to understand what the software actually does. The Way of the Void is directly produced in the sense that we produce the code. Developers often view these tasks as overhead. The written code is productive; there is no writing model or documentation. Nevertheless, in a mature software project, these tasks must be performed.

The traditional life cycle and MDA development cycle, shown in Figure 1, do not differ much from each other. The same phases are defined. One of the main differences is the nature of artifacts created during the development process. Artifacts are formal models, that is, a model that can be understood by computers. The following three models underpin MDA [2].

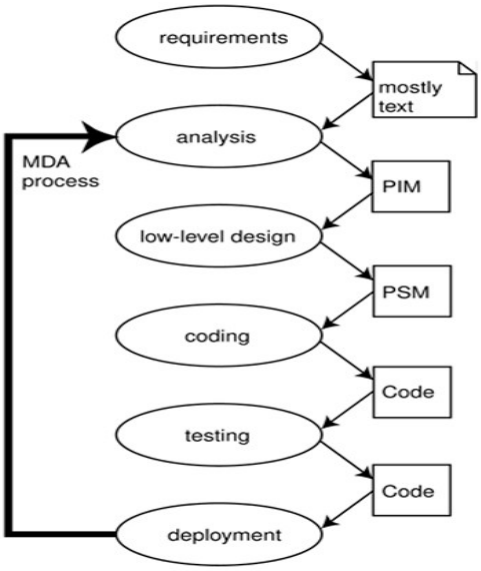

Figure 1. MDA software development life cycle

The first model that underlies MDA is a model with a high level of abstraction that is not dependent on the implementation technology. This is called a platform-independent model (PIM). PIM describes a software system that supports certain types of business. Within the PIM system is modeled in terms of how it best supports the business. Whether the system is implemented on a mainframe with a relational database or an EJB application server does not play any role in PIM.

The second model can be built on the basis of the already developed PIM in one or more models, called the Platform Specific Model (PSM). PSM is designed to point your system in terms of implementation designs available in one particular implementation technology. For example, EJB PSM is a model of the system in terms of EJB structures. It usually contains EJB-specific terms such as "home interface", "entity bean", "session bean", etc. The PSM relational database includes terms such as "table", "column", "foreign key" and so on. It is clear that PSM will only make sense for a developer who has knowledge of a particular platform.

The last model is a code model. It is generated from each PSM that the developer could have. PSM approaches its technology quite closely, this conversion is relatively simple. These three basic models in the MDA are the most important, and around them the entire infrastructure is built.

The MDA pays great attention to the development of PIM (platform independent model). The required PSM (platform-specific model) is generated by converting from PIM to PSM. Of course, someone needs to determine the exact transformation, which is a complex and specialized task. But such a transformation must be defined only once and then can be applied to the development of many systems. The payback of efforts to determine the transformation is great, it can only be done by highly qualified people. Many developers focus on the development of PSM. Because they can work regardless of the details and features of the target platforms, there are many technical details about which they do not

need to worry. These technical details will be automatically added by the PIM transformation into the PSM. This improves performance in two ways.

First, PIM developers have less work to do, because platform-specific details must be designed and written; they are already considered in the definition of transformation. At the PSM and code level, much less code is written, because most of the code is already generated from PIM [2,3].

The second improvement is due to the fact that developers can switch the focus from the code to PIM, thereby paying more attention to solving the business problem. This leads to a system that is much better suited to the needs of end users. End users get better functionality in less time. This increase in performance can only be achieved through the use of tools that fully automate the generation of PSM from PIM

The software industry has some feature that makes it different from most other industries. Every year, and sometimes faster, new technologies are created and become popular (for example, JAVA, Linux, XML, HTML, SOAP, J2EE, .NET, JSP, ASP, Flash, Web services, etc.). For many reasons, many companies must follow these new technologies. First of all, follow the requirements of customers (for example, web interfaces). Finally, it solves real problems (XML for sharing or Java for portability). The vendors of the tool no longer support the old technologies and are focused on the new (UML supersedes OMT).

Software systems rarely live apart. Many systems must interact with other, often existing systems. As a typical example, we have seen that in recent years, many companies are building new web systems. A new final application is launched in a web browser (using various technologies such as HTML, ASP, JSP, etc.), and it needs to retrieve information from existing internal systems. Even when systems are completely built from scratch, they often cover several technologies, sometimes both old and new. For example, when a system uses Enterprise Java Beans (EJB), it must also use relational databases as a storage engine. Over the years, we have learned not to build huge monolithic systems. Instead, we try to create components that do the same job, interacting with each other. This increase (or is even possible) the ability to make changes to the system. Absolutely different components are built using the best technologies for work, but they need to interact with each other. This created the need for interoperability.

MDA solves this problem using bridges. Bridging is the relationship between the PSM and the code, as shown in Figure 2. When PSM is targeted to different platforms, they cannot communicate directly with each other. Either way, we need to transform concepts from one platform into concepts used on another platform. This is what concerns interaction. MDA solves this problem by creating not only PSM, but also the necessary bridges between them. Cross-platform interoperability can be implemented by tools that not only generate PSM, but also bridges between them, and possibly also on other platforms. You can "survive" technological changes while keeping your investments in PIM

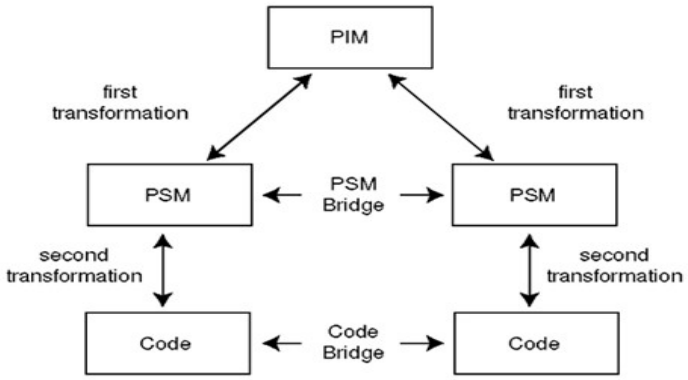

Figure 2. MDA interoperability using bridges

Documentation is a weak link in the software development process. This is often done as a backward thought. Most developers believe that their main task is to create code. Writing documentation during development takes time and slows down the process. It does not support the main task of the developer [4]. The availability of documentation is similar to doing something for the benefit of others, and not for your own sake. There is no incentive to write documentation other than your manager, which tells you what you should do. This is one of the main reasons why the documentation is usually not very good quality. The only people who can check the quality of documentation are the developers who also hate writing documentation. This is also the reason that the documentation is often not updated. Every time you change the code, you need to change the documentation manually. Of course, the developers are wrong. Their task is to develop systems that can later be changed and saved. Despite the feelings of many developers, written documentation is one of their main tasks. Solving this problem at the code level is the ability to generate documentation directly from the source code, ensuring that it is always up-to-date. Documentation is part of the code, not an individual component. This is supported in several programming languages, such as Eiffel and Java. This solution, however, solves only the problem with the documentation at a low level. The higher-level documentation (text and diagrams) still needs to be written manually over and over again. Given the complexity of systems that have built documentation at a higher level of abstraction, it is absolutely necessary.

Therefore, working with the life cycle of MDA, developers can focus on PIM, which is at a higher level of abstraction than code. PIM is used to create a PSM, which in turn is used to generate code. The model is an exact representation of the code. Thus, PIM performs the high-level documentation function, which is necessary for any software system. The only difference is that PIM is not left after writing. Changes made to the system will ultimately be done by changing the PIM and restoring the PSM and code. In practice today, many changes are made to the PSM, and the code is recovered from there. Good tools, however, will be able to communicate between PIM and PSM, even when there are changes in the PSM. Thus, the changes in the PSM will be reflected in the PIM, and the documentation at the high level will remain in accordance with the actual code. Naturally, in the

MDA approach, documentation will be available at a high level of abstraction. Even at this level, it remains necessary to record additional information that cannot be written to PIM. This includes, for example, the reasoning for the choice that was made during the development of PIM.

References

1 Kleppe A. G. et al. MDA explained: the model driven architecture: practice and promise [Text] / – Addison-Wesley Professional, 2003.
2 Mellor S. J. et al. MDA distilled: principles of model-driven architecture [Text] / Addison-Wesley Professional, 2004.
3 Nguyen T. H., Dang D. H., Nguyen Q. T. On Analyzing Rule-Dependencies to Generate Test Cases for Model Transformations [Text] / 2019 11th International Conference on Knowledge and Systems Engineering (KSE). – IEEE, 2019. – C. 1-6.
4 Alam M. M., Breu R., Breu M. Model driven security for Web services (MDS4WS) [Text] / 8th International Multitopic Conference, 2004. Proceedings of INMIC 2004. – IEEE, 2004. – C. 498-505.