Technology for application family creation based on domain analysis

Gudoshnikova Anna
Chair of informatics
St.Petersburg State University (SPbSU)
St.Petersburg, Russia
Email: gudoshnikova.anna@gmail.com

Yurii Litvinov
Software Engineering chair
St.Petersburg State University (SPbSU)
St.Petersburg, Russia
Email: y.litvinov@spbu.ru

Abstract—The theme of code reuse in software development is hot-button issue. Sometimes it is hard to find out what exactly we need to reuse in isolation of context. However, there is an opportunity to narrow the context problem, if applications in one given domain are considered. Hence, the problem of domain analysis arises. On the other hand, there is metaCASE-techonology for application code generation with diagrams, which describe the given application. The main objective of this article is to present the technology for application family creation, which connects the metaCASE-techonology and domain analysis. We propose to use feature diagrams to describe variability in a domain and then create domain-specific visual language that allows to connect and configure existing feature implementations thus producing an application. This technology supposed to be especially useful for software product lines.

Index Terms—domain analysis; metaCASE-technology; domain-specific language; application family

Introduction

The term "reuse" in software engineering closely associated with context. Reuse objects can be programs, parts of programs, specifications, requirements, architectures, test plans, etc. Reuse of one object leads to reuse of another object. This means, there is a need to reuse something more than just code, i.e. there is a call for increasing the abstraction level. It is commonly supposed that reuse, as some kind of activity, can be divided into groups according to what should be reused: components, process for gaining the product, the technology or knowledge. At all accounts any reuse object cannot be discussed without environment, where the given object exists. Hence, the context problem is still remains.

However, if we reuse objects in one domain, the context issue may be narrowed. The product line implies that there is a common part, it can be: (1) architecture, (2) components, (3) algorithms, (4) methods, etc. — and this part exists in the same context. This fact facilitates the reuse problem. Consequently, the common part must be reused.

Gathering information about the domain is the crucial step in the whole process of software development.

Nowadays, applications in one domain are often designed independently; this approach leads to increase of development time and cost. Usually such applications have similar functionality, so the reuse problem moves to the forefront in an attempt to speed up the development and to decrease the cost for systems in one domain. The reuse process in one domain supposes the necessity of the domain analysis activity. At present domain analysis in software life cycle is performed in informal way. There are some domain analysis tools, but such tools are not integrated with development tools. As the result of the domain analysis activity some diagrams just are put up on the board, and do not take part in following process of software design. The risk of incorrect understanding of domain-dependent knowledge increases. Therefore, many peculiarities of the domain may be missed in development process because of the factor of human error. This fact may lead to development of the product which do not satisfies requirements at all. Hence, there emerged a need for a tool, in which domain analysis activity would play a vital role in software development process, i.e. based on this activity would be possible to generate some design model, so developers and other process actors could rely on this model. At the present day there is no tool that could allow to solve this problem.

One possible solution for this problem is the use of domain analysis tool in model-driven development, or, more precisely, domain-specific modelling. Domain-specific approach uses visual languages to specify system under development, but, contrary to general model-driven approach which uses general-purpose visual languages like UML, domain-specific languages are tailored specifically for given domain or a set of problems. Existing studies [1], [2], [3], [4] show that due to closeness to a problem domain and the ability to generate complete application by visual models domain-specific languages boost development productivity by 3 to 10 times compared to general-purpose languages. It is clear that developing a tool for domain-specific language "from scratch" for each domain will be prohibitively costly, so special systems are used that allow to declaratively specify syntax of a language and to automatically generate such tools as visual editor, source

code generators, constraints checkers and so on. Such systems are called DSM platforms, most known of these is MetaEdit+ [5], [6], [7], Eclipse GMP [8], [9], Microsoft Modeling SDK [10].

Main idea of domain-specific modeling is to use a number of visual languages in one tool to develop a complete system. Every language can provide a different point of view on a system. We propose to exploit this idea to automatically produce useful artefacts from the results of domain analysis thus seamlessly integrating this phase into development process. For that, we will use specific visual language to perform domain analysis and to build domain model, language simple enough to be useful to analysts and domain experts who do not necessarily possess programming skills. Then, using this domain model, we will generate actual domain-specific language that will allow to configure various existing pre-built components and integrate them to generate a working application. As we will see, this language will also typically be very simple so that non-programmers can use it. The only real coding in the proposed approach occurs when creating components from which applications will be built, but for product lines these components will already exist anyway, as they will in a case when a team develops many applications in one domain for some time. Not all steps in proposed approach are fully automatic, as a visual language needs tailoring after generation from domain model — we still need to manually specify shapes of its elements (to be familiar for domain experts) and configure properties which depend on existing components and can not be derived from domain model. It is also possible that generated application will need tailoring by hand, but generation can significantly lower the effort needed to create application.

Main contribution of this research-in-progress paper is a novel approach to product line development and assets reuse. Also an implementation of technology which uses this approach is presented. Our technology is based on QReal DSM platform [11], an open source tool developed by Software Engineering chair of St. Petersburg State University[1]. An evaluation of proposed approach is also presented, but on a rather simple problem, so a much wider evaluation is needed for this study to be considered complete, such as the applicability of this approach to complex real-life situations and determining actual productivity boost on real-life problems.

The rest of this paper is structured as follows: in section I most important terminology for domain analysis is given, also related works are considered. In section II we present our method and its implementation as development platform, in section III an example of application of our approach is given, we will consider a family of Android gamepads for remote control of various robot models. Section IV concludes the paper.

[1]GitHub repository and home page of QReal project, URL: https://github.com/qreal/qreal (03.04.2016)

I. Domain analysis approaches

There is no any clear and long-standing definition of the term "domain analysis". Almost all papers, in which this term is considered, go back to 80s-90s of the twentieth century. It was then that scientists, taking into account rapidly growing technologies, were thinking about global reuse. Always projects are developing for concrete user needs, so then the term "domain" took the definition. Domain is the field of expertise, problems in which the software intend to solve. According to Rugaber [1], the domain is described in terms of glossary, some assumptions, architecture approach and literature.

Then the question arise, how we need to analyze the domain for acquiring the necessary information. At present, the information gathering into knowledge bases is understood under the term "domain analysis". Although, Prieto-Diaz [2] confirms that domain analysis is an activity, which take place before system analysis and its output is used for system analysis to the same degree as system analysis's output is used for system design. There are other definitions of the term "domain analysis". Ferre [3] has presented definitions, such as: (1) the process of identification, organization and presenting the relevant information of a given domain, (2) the process, in which the customer's knowledge are identified, concretized and systemized. The relevant information of the domain should be presented in objective, readily available way, such way is called "domain model". Mernik [4] specifies, that the domain model include not only glossary, but also must describe commonalities and variabilities of terms. Such model should precisely set bounds of the domain, i.e. clear and exact characterize a range of questions, which are considered in the domain. Term variabilities allow to define exactly, what information must be specified in concrete system implementation. Term commonalities are used for defining a set of shared operations between different applications. Implementing commonalities and adding the gained model with information, which can be specified in instance of the concrete system, a set of different system can be obtained based only one common model. In such manner, based only one domain model can be implemented the set of different systems in given domain. Taking into account definitions above mentioned, we can conclude that domain analysis is the activity forward system analysis, which goal is providing the domain model.

As stated above, at present in many software companies the term "domain analysis" is understood as information gathering into some knowledge bases, but it is obviously that there are disadvantages of this approach. It may lead to incomplete glossary, absence of agreements about understanding some terms in the domain, so any misunderstanding of domain can result an improper product. Therefore, several dozens of years ago were introduced some formal approaches for domain analysis. Here will be mentioned some of them. It is worth noting that main

objective any domain analysis approach is to gain the domain model.

Despite different understanding the term "domain analysis" Arango [] showed that all formal domain analysis methods follow the general process for obtain the domain model. This process includes next stages: (1) domain characterization, (2) data collection, (3) data analysis, (4) classification and finally (5) evaluation of domain model. There are following domain analysis approaches: 1) DARE (Domain Analysis and Reuse Environment) []. The crucial idea of this method is creating domain book. This book include the universal architecture and library of reusable components. 2) DSSA (Domain-Specific Software Architectures) []. Given approach allows to create a domain glossary with the aid of use case analysis. 3) ODE (Ontology-based Domain Engineering) [] This approach connects the ontology idea with object-oriented approach. Ontology includes terms and their connections, definitions, properties and constraints. Library of objects is built based on mapping ontology with object-oriented entities. 4) FODA (Feature-Oriented Domain Analysis)[]. This method has get popularity among scientists in the research area because of its simplicity for non-programmers. The main idea of this approach is creating feature model. This model describes functionality, which the future product should possess. Such model must note, what features are compulsory for implement in any instance of application in a given domain, what features must be implemented but there is some alternative between them, and present features, which may be implemented but not compulsory. This model can be easily built by expert in the domain.

Concerning product line creating with the aid of using domain model, Estublier[] presented approach, which is based on some aspects and requirements. These entities were proposed by authors. Such approach based on MDE methodology. Domain model is considered as metamodel, which is described on MOF or UML. There is an interpreter, which translates each term in metamodel into Java-class, and concrete models — into instances of these classes. Domain model is accompanied with feature model, which include some external behavior of the system. Authors use aspect-oriented techniques for feature implementing and following their mapping with terms in domain model. Consequently, there is a close interaction between domain modeling and feature modeling. It seems that such approach is a bit complicated for non-programmers. In addition, there is no any industrial use of this method, but it is worth noting that authors describe appliance in this article [].

II. Proposed approach

In our approach we will use Feature-Oriented Domain Analysis (FODA) method to perform domain analysis and to create feature models. For this we will use visual editor that implements feature diagrams and is easy enough to use that domain experts can use it without problems.

Then, when feature models are ready, each feature is implemented as reusable and configurable component on selected implementation language (C#, C++, Java and so on) and feature library is formed as a collection of such components. This process requires qualified programmers and requires more effort than to simply create one application, but it allows to reuse features from feature library to create as many applications as needed. Also, this process is scalable, so we may add new features into feature library later, thus allowing to create more complex applications. At this stage of development domain experts shall work with programmers, and they shall use feature diagrams as an input for creation of feature library to simplify matching between features and components in feature library.

Next step is to create domain-specific language that allows to combine and configure features from feature library to implement applications in given domain. This is where our approach differs from common reuse strategies. Naive approach would be to generate an application directly from feature diagram, somehow marking features that shall be included into application, and it actually works fine when domain variability is low []. But more common is the situation when features themselves have properties that allow to configure them, those properties can have different types. Also, components may be related to each other in different ways, be used in configuration of one another, or some of their properties may be meaningless in absence or presence of other feature. Those rules may be implemented implicitly in application generator and require that programmers will always observe them, but we propose that these rules will be captured explicitly by dedicated domain-specific language. Such language may make models that do not observe those rules syntactically incorrect, and it will greatly reduce the possibility of human error and reduce knowledge required to efficiently use programming system.

By using DSM platforms one can relatively quickly create domain-specific language that will capture domain knowledge, but we already have feature diagram, so we actually can generate the language using it. Generator takes feature diagram as input and produces metamodel of a language. Metamodel is a visual model of a language syntax, that can be opened and edited in yet another visual editor that is part of DSM platform, this editor is called metaeditor. Features from feature diagram become entities in metamodel, this metamodel is then edited to provide shape and a list of properties for each entity. Any vector image can play the role of shape, so the best practise is to select shape that is similar to a feature it depicts. For example, if an application can have buttons, "button" becomes entity in domain-specific language and looks like a button on a diagram. The same happens with properties — for each feature they are added in metaeditor to corresponding language entity with respect to feature library that actually implements this feature and uses the
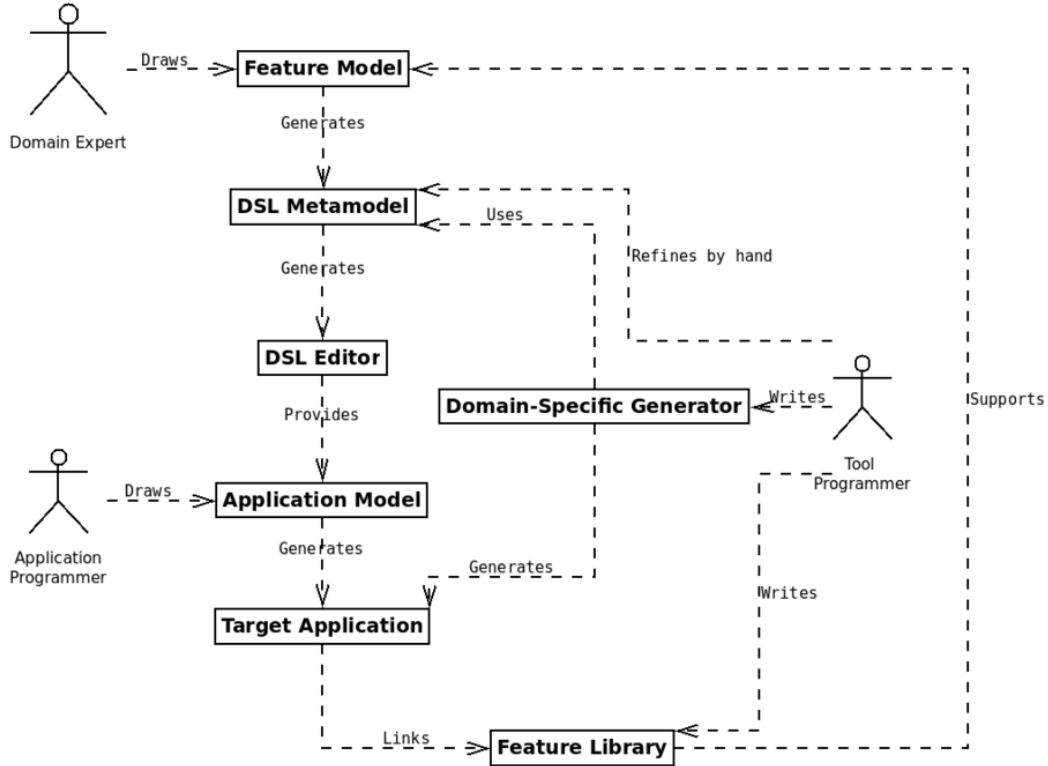
Domain Expert — Draws → Feature Model ← — — — — — — — — —

Feature Model — Generates → DSL Metamodel ← Uses — —

Refines by hand

DSL Metamodel — Generates → DSL Editor

DSL Editor — Provides → Application Model

Domain-Specific Generator ← Writes — Tool Programmer

Supports

Application Programmer — Draws → Application Model

Application Model — Generates → Target Application

Target Application — Generates — —

Writes

Target Application — Links → Feature Library ← — — — — — —

Figure 1: Relations between artefacts and roles in proposed approach to domain components reuse.

property to configure it. Properties have name, type and default value. On this step it is also possible to define some constraints on a metamodel that will be checked when model will be edited. If some constraints are violated, user will immediately receive warning, which makes errors in a target application even less likely to occur.

On a next step we use editor generator of the DSM platform to create visual editor for our newly created language. This step if fully automated, and when an editor is generated and loaded into DSM platform, we can use it to create diagrams that specify target applications.

The next thing we need is to generate actual code on target textual language that will call feature library and glue features together. For this we shall return to metamodel level and define generation rules for metamodel. This step is performed only once for a given domain after the feature library and metamodel are finished, and then the same generator is used for each application created by using of the technology. Recommendations for development of domain-specific generator are well-known in DSM literature (for example, [7]): it is the best to write first application by hand, then draw a model that is supposed to be generated into this application, then find the places in handwritten application that shall be parameterised by information from model and let the generator replace such handwritten parts with data from model. This process is continued until handwritten application becomes a template that is filled by generator with information taken from model. Handwritten application and, consequently, a generator shall extensively use feature library to minimize the amount of code that is generated directly, in ideal case generator shall produce merely a glue code that binds components from feature library together.

After all steps above are finished we have feature library, visual editor for simple domain-specific language that allows to describe how features are combined and configured in a concrete application, and a generator that automatically produces complete application by a model in domain-specific language using feature library as domain-specific runtime [7]. Now we may create as many applications as we wish by just drawing models and automatically generate complete executable code. Theoretically. Of course, in practise there will always be a need to modify feature diagram, to extend feature library and, consequently, domain-specific language metamodel, modify generator and even to make some changes in generated code, there is no silver bullet. But we believe that our approach can provide better separation of concerns, provides better utilization of domain experts knowledge and expertise among a team. Summary of a process described above and relation between various tools and roles of developers is provided on Fig. 1.

This approach was implemented in a technology based on QReal DSM platform. QReal became an enabler technology because it provides easy and effective way to create visual editor for domain-specific languages that allows to create fully functional editor in less than an hour. It has visual metaeditor, visual constraints definition tool, visual shape editor and a C++ library that allows to quickly specify generation rules. Feature diagram editor and generator that creates metamodel by feature diagrams were both implemented as plugins to QReal core. Note that feature diagram language is itself domain-specific language for the domain of domain analysis, so it was implemented using QReal metaeditor. The same metaeditor (including shape editor and constraints editor) is then used to tailor the generated metamodel of domain-specific language. Then metaeditor generator is used to generate yet another plugin to QReal that provides visual editor for created language. Then the generator is implemented by hand on C++ with Qt library[2] using generator creation library included in QReal. Then it is possible to create special distribution of QReal (using Qt Installer framework[3]) that includes only QReal core, editors for feature diagrams (at this point they are needed only as reference) and domain-specific language, generator and feature library, thus forming a complete technology that can be used to generate target applications.

III. Evaluation

For efficiency demonstrating of proposed above approach there was implemented a model application for remote control of various robot models — "Joystick". The main substantiation for implementing such application is that controlling different robot models requires different control elements. For example, one model can be controlled with only two pads, but another — with one pad and two buttons. Such application was implemented in C# for Windows Phone platform. Screenshots of this simple application are presented on Fig. 2.

As mentioned above, it was used QReal as DSM tool. There was implemented a visual language for describing feature models. Appropriate feature model for application family "Joystick" is proposed on Fig. 3.

[2]Qt library home page, URL: http://www.qt.io/ (03.04.2016)
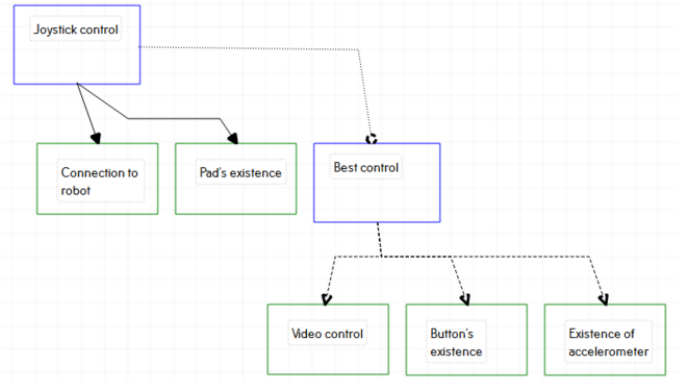[3]Qt Installer Framework home page, URL: https://wiki.qt.io/Qt-Installer-Framework (03.04.2016)

Figure 3: Feature model for "Joystick" application family.

Based on this feature model there was generated metamodel for future visual language, which is required for building different models for different configurations. Generated metamodel is presented on Fig. 4. As it can be seen, metamodel is very simple. At this stage we can propose that entities, such as "buttons" and "pads" may have a property "Quantity". In addition, we can set forms for these entities, which will be shown in visual language.
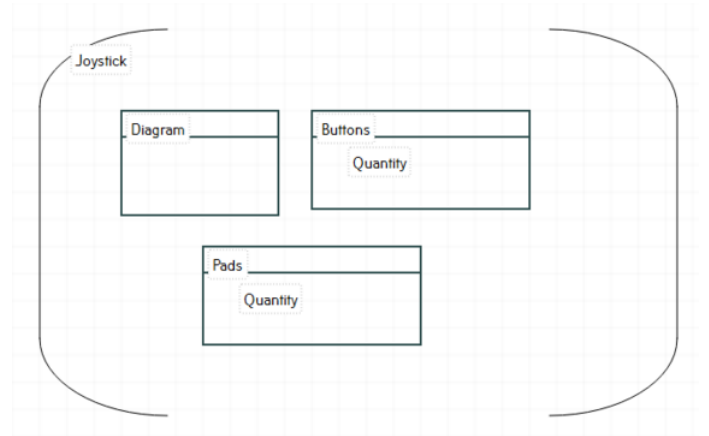

Figure 4: Metamodel of visual language for "Joystick" application family.

Then with resources of QReal tool was generated a visual language. Example of generated visual language is demonstrated on Fig. 5. It can be seen that in visual language editor we can specify property "Quantity", explicitly pointing out concrete number of pads.
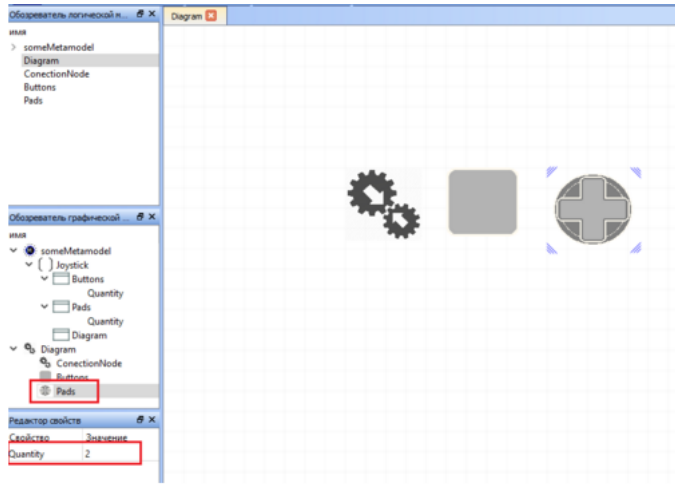
Figure 2: Screenshots of "Joystick" application.

Figure 5: Visual language for "Joystick" application family.

To tell the truth, example is quite simple for demonstrating extensive possibilities proposed above approach. At present there is no rigorous evaluation of the offered process. It is worth noting that it is just an idea, there is no yet cohesive and consistent technology for creating application family based on domain analysis. Therefore, this work requires more detailed explorations.

IV. Conclusion

The problem of not using domain analysis result for further generating some entities for software development process was stated.There were considered some formal domain analysis approaches and we concluded that creation of feature diagrams is the most elegant decision for domain analysis that can be conducted by domain expert, i.e. non-programmer, may be in collaboration with system analysts. Moreover, there was discussed one of the possible solutions, which is presented by Estublier, we specify some problems of such method. We suggested our own approach for creation application family in one domain based on domain analysis. Also, there was some evaluation of this approach, where we pointed out that this example remains many questions because of its simplicity. Finally, this research requires more detailed explorations.

References

[1] Tolvanen J.-p., Kelly S. Model-Driven Development Challenges and Solutions // Modelsward 2016. 2016. P. 711–719.

[2] Baker P., Loh S., Weil F. Model-driven engineering in a large industrial context — Motorola case study // MoDELS'05: Proceedings of the 8th international conference on Model Driven Engineering Languages and Systems. Berlin: Springer, 2005. P. 476–491.

[3] A software engineering experiment in software component generation / R. Kieburtz, L. McKinney, J. Bell et al. // Proceedings of the 18th international conference on Software engineering. Washington, DC, USA: IEEE Computer Society, 1996. P. 542–552.

[4] Kelly S., Tolvanen J.-P. Visual domain-specific modeling: Benefits and experiences of using metaCASE tools // International Workshop on Model Engineering, at ECOOP. 2000. URL: http://dsmforum.org/papers/Visual_domain-specific_modelling.pdf.

[5] Tolvanen J.-P., Pohjonen R., Kelly S. Advanced tooling for domain-specific modeling: MetaEdit+ // Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling (DSM'07). 2007. URL: http://www.dsmforum.org/events/DSM07/papers/tolvanen.pdf.

[6] Tolvanen J.-P.and Kelly S. MetaEdit+: defining and using integrated domain-specific modeling languages // Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications / ACM. New York, NY, USA: ACM, 2009. P. 819–820.

[7] Kelly S., Tolvanen J.-P. Domain-specific modeling: enabling full code generation. Hoboken, New Jersey, USA: Wiley-IEEE Computer Society Press, 2008. P. 444.

[8] Gronback R. Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Stoughton, Massachusetts, USA: Addison-Wesley, 2009. P. 736.

[9] Viyovic V., Maksimovic M., Perisic B. Sirius: A rapid development of DSM graphical editor // IEEE 18th International Conference on Intelligent Engineering Systems INES 2014. Los Alamitos, CA, USA: IEEE Computer Society, 2014. P. 233–238.

[10] Domain-specific development with Visual Studio DSL Tools / S. Cook, G. Jones, S. Kent et al. Crawfordsville, Indiana, USA: Addison-Wesley, 2007. P. 576.

[11] QReal DSM platform-An Environment for Creation of Specific Visual IDEs / A. Kuzenkova, A. Deripaska, T. Bryksin et al. // ENASE 2013 — Proceedings of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering. Setubal, Portugal: SciTePress, 2013. P. 205–211.