# Advantages of UML-based object-oriented system development

Saša Dešić, Darko Gvozdanović, Mario Kušek*, Darko Huljenić
Ericsson Nikola Tesla d.d.
Krapinska 45, HR-10000 Zagreb, Croatia
tel.: +385 (0)1 365-3857 , faks: +385 (0)1 365-3548, email: {sasa.desic, darko.gvozdanovic, darko.huljenic}@etk.ericsson.se

*Department of Telecommunications
Faculty of Electrical Engineering and Computing
University of Zagreb
Unska 3, HR-10000 Zagreb, Croatia
tel.: +385 (0)1 612-9748 , faks: +385 (0)1 612-9802, email: mario.kusek@fer.hr

**Abstract - UML stands for Unified Modelling Language, a general-purpose notational language for specifying and visualising complex systems, especially for large, object-oriented projects. This article considers usability of UML in software projects. It is especially interesting to look for impact that UML has on newcomers in the world of object-oriented software development. An experiment with two groups of students (one group was trained for UML) was made. Groups' goal was to develop solution for particular software system. UML's advantages and disadvantages are also commented according to user's level of knowledge, application type (e.g. protocol oriented or GUI oriented applications) and requirements detail level.**

## I. INTRODUCTION

The good news for software developers is that world development and modern style of living depends increasingly on software. Software-intensive systems that technology makes possible and society demands are expanding in size, complexity, distribution and importance. But, expansion of these systems pushes the limits of what software industry know how to develop. The final result is that building and maintaining software is hard and getting harder.

Different software development projects fail in different ways, but it is possible to extract common symptoms. Some of them are: inaccurate understanding end-user needs; inability to deal with changing requirements; software that's hard to maintain or extend; late discovery of serious project flaws. If symptoms are analysed, root causes could be find: ad hoc requirements management; ambiguous and imprecise communications; overwhelming complexity; undetected inconsistencies in requirements, designs and implementations; uncontrolled change propagation; insufficient testing.

Some of problems and their root causes could be resolved by implementing more rigorous development process. Also, deployment of notification language like UML will help communication between all participants in development process. This is just one of the reasons for UML usage in process of software development. This article analyses aspects of UML using in development process.

The second section explains root, causes and purposes of UML. UML diagrams are explained briefly. Next section introduces one software development process that utilises UML. Examples and aspects of UML utilisation are shown in fourth section. In the fifth section, experiments with UML and inexperienced developers are described. Final section summarises the results.

## II. UML

The unified modelling language (UML) is graphical language for visualising, specifying, constructing, and documenting software-intensive systems. The UML provides a standard way to write a system's blueprints, covering conceptual things, classes written in a specific programming language, database schemas, and reusable software components. UML is standard notation, which is used by anyone involved in production, deployment, and maintenance of software.

UML includes nine diagrams for describing system:
- Class diagram describes set of classes, interfaces, and their relationships. It shows static design view of a system. This diagram is very useful in modelling object-oriented systems.
- Object diagram shows set of objects and snapshots of instances of the things found in class diagrams.
- Use case diagram shows a set of use cases, actors and their relationships. This diagram is especially important in organising and modelling the behaviour of a system.
- Sequence and collaboration diagrams are interaction diagrams, which describe interaction between objects. They show their relationships including messages between objects. Interaction diagrams explain dynamic view of the system. A sequence diagram emphasises the time ordering of messages. A collaboration diagram emphasises the structural organisation of objects in interaction. Both diagrams are isomorphic, which means that sequence diagram can be transformed into collaboration and vice versa.
- Statechart diagram shows a state machine consisting of states, transitions events, and activities. It addresses dynamic view of system. This diagram is very important in modelling behaviour.
- Activity diagram is special kind of statechart diagram. It emphasises a flow from activity to activity within system. It is very useful in finding concurrent activities in system.
- Component diagram describes organisation and dependencies among a set of components. It is related to class diagrams in a way of mapping component to one or more classes, interfaces, or collaborations.

- Deployment diagram explains configuration of run-time processing nodes and components on them. It shows static deployment view of architecture.

## III. DEVELOPMENT PROCESS

The UML is modelling language, not a method. But for successful project, modelling language is not enough. There are several developing methods existing worldwide (e.g. Extreme programming, Feature Driven Programming). Creators of UML made Rational Unified Process (RUP) and it utilises UML the most. Best practices of different projects are incorporated in it. This section briefly explains the Rational Unified Process.

The Rational Unified Process is iterative and incremental development process (Figure 1.). Software, through process, is carefully built step by step, functionality by functionality. After every iteration, subset of functionality will be designed, implemented and tested.

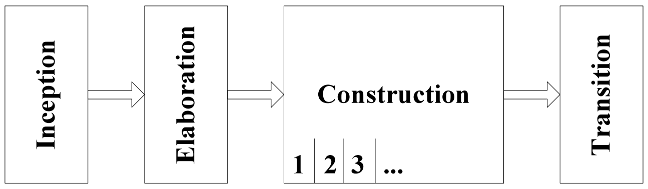

Figure 1. The Rational Unified Process

During **inception**, first meetings about goal and the scope of project are held. Also, business case (roughly how much it will cost and how much it will bring in) is created.

In **elaboration**, more details about requirements and technologies involved must be caught. Also, high-level analysis and design for creating baseline architecture is done. In this phase, plan for construction phase is also created.

The **construction** phase consists of many iterations, in which each iteration builds production-quality software, tested and integrated that satisfies a subset of requirements of the project. The delivery may be external, to early users, or purely internal. Each iteration contains all the usual life-cycle phases of analysis, design, implementation and testing.

The **transition** includes beta testing, performance tuning and user training. Optimisation reduces the clarity and extensibility of the system in order to improve performance.

## IV. UML IN PRACTICE

Enthusiastic approach toward UML has been questioned in a real world through an experiment. Two software systems had to be designed using UML along with the appropriate tool. Results should have shown UML usability regarding different types of systems and different required detail level. Therefore, two systems were chosen:

- agent system - requirement are fairly general and architecture is accented,
- communication application (implementing specific protocol) – there is detailed requirements documentation and accent is on behaviour.

After having those systems designed, conclusion that we already had remained: UML is superior in early phases of development. Requirement analysis is heavily supported, but two types of diagrams distinct themselves. Use-case and sequence diagrams provide everything needed for capturing all the systems features, yet retaining simplicity needed for customer to understand designed behaviour. Misunderstanding between designers and customers is excluded if you use UML, and misunderstanding itself usually leads projects into failure. Negotiation with customers (they often have a tendency to change their mind), if not good design praxis itself, forces us to design system to be resilient to moderate change in requests. Another good side of having use-cases is that they are valuable source for creating test-case scenarios for testing and validating partly or fully implemented systems.

One could not expect miracles from use-case diagrams. Maybe most difficult task in design is to “draw right” use-cases. Use-case diagrams give means to express ideas, but ideas come from great experience. Less experienced designers maybe would not understand reasons behind particular design made by senior designer (they will notice its simplicity and beauty though), but they will understand it and surely be able to upgrade on it. In fact UML, inheriting only the best features of many modelling languages, can really help while acquiring object-oriented way of thinking. “I can’t express it with UML” in most cases means that there is something wrong with your design (regarding object-orientation). So, using UML is very helpful to all categories of designers.

Another important issue has been tackled by UML. Parallel design of different parts of a system is almost obligatory except for very simplest ones. Precondition for that is proper system division, which comprises two things:

- definition of relatively independent subsystems, having appropriate functionality,
- ensuring full collaboration of those subsystems, once they are designed.

The way that UML addresses these questions is by packages and interfaces. There are no package diagrams but one can use class diagrams for that purpose. In that way, not only implementation but also design is hidden behind interfaces, facilitating later changes and maintenance. It is even possible to develop particular parts of the system using different tools (without any UML), and than reverse-engineer it back to UML design (Figure 2). This particularly refers to GUI design. As redesign or adding new functionality is often the case (it’s not always about brand new system), using legacy code can make substantial savings.

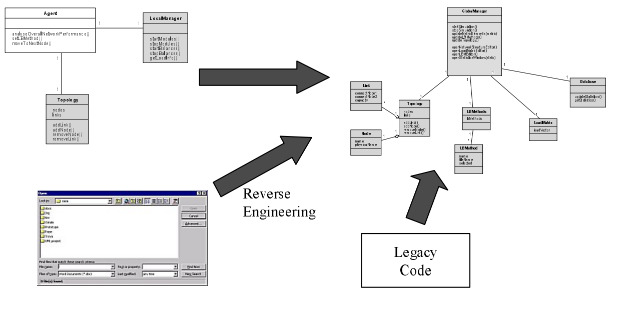

Figure 2. Reverse engineering of GUI to UML diagram

It was obvious that UML handles architectural problems well. Defining behavior is it's less good part. There is whole bunch of diagrams (sequence, collaboration, activity and state-chart diagrams), but designer can use it only for it's own needs. They don't contribute to the automatic code generators at all. So, one can say that a lot of trouble about capturing behavior of mentioned communication application is worthless. It isn't just like this, but designers would definitely like if there had been some reflection of the captured behavior in the code. Therefore, big step ahead is UML to SDL (Specification and Description Language [4]) converter offered by some vendors. SDL is suitable for in-depth system design and verification. Also, SDL can be successfully used to express system dynamic behavior. So, by using translator, a lot of information from UML is kept and transferred into SDL. Consequently, visual design along with all the possibilities of system verification and validation offered by the tools remains available to designers.

## V. EXPERIMENTS WITH INEXPERIENCED DEVELOPERS

A lot of articles about UML describe its tremendous features but little of them show exact results from experiments with UML in real projects. It is hard to quantitatively express advantages of UML comparing to common way of software development. To clear picture about UML usefulness, an experiment with group of students were made.

Students were separated into two groups. First group (UML group) were listened presentation on UML. For that time, second group (Non-UML group), which was separated into three smaller groups, tried to design three small systems: web service for exam appliance, cache machine and automatic door opening system. They had not any information about UML and they were left to design systems with their previous knowledge and sense. Here, it's key issue that all students were actually with no experience with object-oriented system design and development.

It is hard to quantitatively measure design and to compare progresses of two different groups. But, as a one of the elements, amount of created diagrams and materials can be used. The non-UML group has created a few roughly descriptions of the system operating and flow diagrams (Figure 3). Most of the time they were trying to catch up with internal structure and architecture of the system. UML group made more. They mostly created Use Case, Sequence (Figure 4), and Activity diagrams. They even created Class diagrams that show classes, methods and attributes. According to an amount (quantity) of created documents, it is obvious that UML has extremely helped in design efforts (especially to capture requirements and to see overall system architecture).

Implementation as final goal of every development process requests a lot of details to be defined. A level of details can also be a parameter to compare two groups and their work. The non-UML group and its designs are on very high level of abstraction and with little of details. A programmer, who is responsible for implementation, will be very sad with such kind of input documents. On other side, UML group with class diagrams containing a lot of details (attributes and methods) has a very good basic for implementation. Other created documents will be also useful for system visualisation and understanding how it works and that will certainly help programmer.

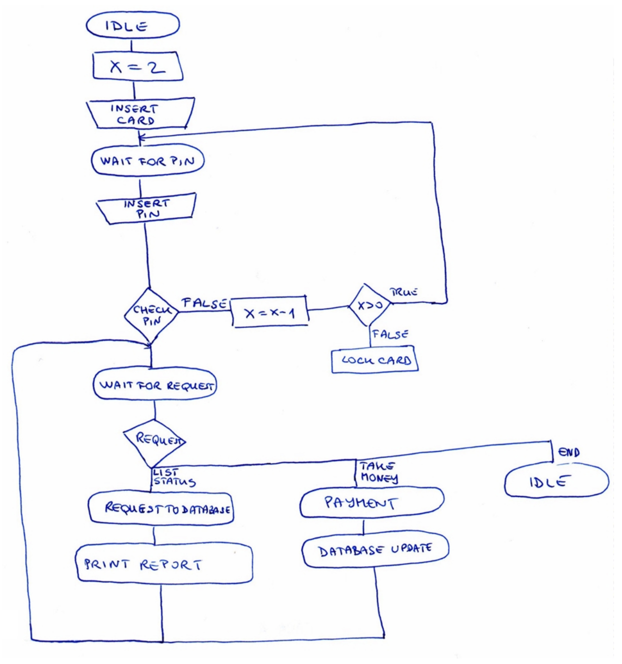

Figure 3. Flow diagram created by Non-UML group

From non-UML documents, it is obvious that group were a lit bit confused and disoriented. Contrary, UML group was leaded with UML diagrams and became more concentrated and effective. Logical flow of diagram usage helps sustaining main design effort in right direction.

To determine system classes and their methods and attributes, the non-UML group has been used only power of their brains. In a bigger systems that becomes practically impossible to handle. The UML has used powerful tools like a sequence and class diagrams (Figure 5). On that way, developers are driven to find appropriate solution for system classes and communication among them.

In early stages of development process, it is very important to examine different possibilities and irregularities in the system. As in the previous case, for the non-UML group that was only a mental process. But, analysis with use case and sequence diagrams can help to find, develop and explain system states where more then one option is possible. Unfortunately, UML group didn't pay attention on that problems and options. That is probably happened because this is their first project in UML. Second reason can be lack of time, because both of groups were time limited.

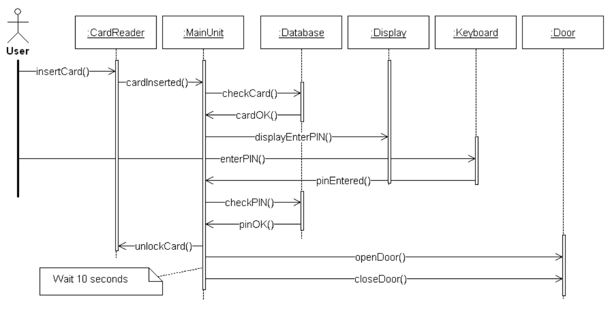

Figure 4. Sequence diagram created by UML group

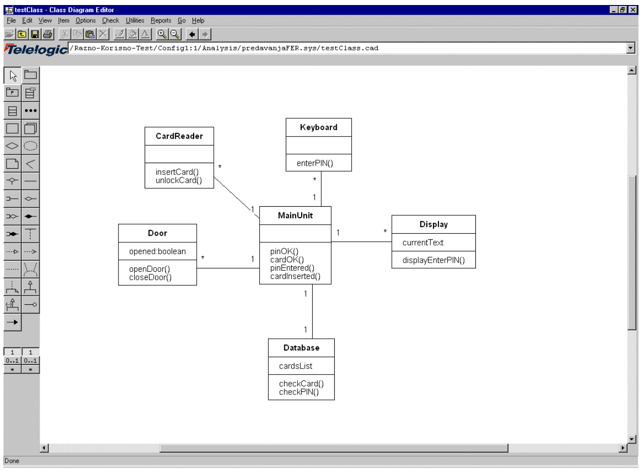

Figure 5. Class diagram created with UML tool

To capture requirements and divide system into smaller parts isn't such a big problem if we have small system to design. The non-UML group was lucky and avoided big trouble. But, UML gives opportunity to handle a huge system by analysing them and dividing them into several packages. Package diagrams also were not used.

The non-UML group has used flow diagrams that help for system visualisation. It is very useful method that makes development process easier and more organise. All UML diagrams also help to visualise system but not only its behaviour like flow diagrams but from all system's aspects. It is especially handy when some UML tool is used.

## CONCLUSION

It can be concluded that UML has a lot of power to offer. New projects have better chances to survive if UML is used. The experiment with students shows that UML has a very strong impact on newcomers and it can strongly increase their working/designing capabilities.

Usage of UML tools can additionally improve software design process. With those tools documentation process will be included into development process because documentation is created during design time.

Using UML in practice proved to be good. Requirement analysis and architectural design are great to do with UML. Additionally, advent of UML to SDL translators gave strength to process of systems' dynamic structure design, too.

## REFERENCES

[1] G. Booch, J.Rumbaugh, I. Jacobson, *The Unified Modeling Language Use Guide,* Addison-Wesley, 1999.

[2] M. Fowler, Kendall Scott, *UML Distilled*, 2nd edition Addison-Wesley, 2000.

[3] P. Kruchten, *The Rational Unified Process*, 2nd edition Addison-Wesley, 2000.

[4] J. Ellsberger, D. Hogrefe, A. Sarma, *SDL Formal Object-oriented Language for Communicating Systems*, Prentice Hall 2000.

[5] O. Laitenberger, C. Atkinson, M. Sclich, K. El Amam, *An Experimental Comparison of Reading Techniques for Defect Detection in UML Design Documentation*, Fraunhofer, Kaiserslautern, 1999.