Special Aspects of Component Based Software Development

József Tick

tick@bmf.hu

Abstract: The Component Based Software Development (CBSD) is the new answer in the field of Software Engineering to the challenge of creating large software systems.Systems developed this way are more stabil, they have better quality, they are cheaper and meet better the requirements. This paper discusses the role and importance of Component Based Software Development in the evolution of Software Methodology, and it analyzes the special aspects of Software Engineering training in Higher Education..

Keywords: Software Engineering, Component Based Software Development

1 Introduction

The last practically 60 years in the software development talks about the history of challenges and responses to these challenges, in this specific case the programming praradigms as well as development methodologies. A demand for the development of larger and larger and more and more complex software required software handling at a higher and higher abstraction level. The significant expansion of hardware and software resources made more and more complex development solutions possible. The newest step in this improvement is the Component Based Software Development (CBSD), and its methodology base, the Component Based Software Engineering (CBSE).

Let's make the principle of CBSD clear via an example. We need a computer that meets our special requirenments. We can select from hundreds of PC cards that meet our requirements browsing the catalogues. Interfaces are well defined, each modul keeps the communication protocols, and functionality is clearly defined. We do not need a soldering-iron, we do not need ICs, we can build from previously developed, already tested, good quality, larger components. CBSD forges to do something similar. Software systems can be developed by assembling previously developed, complex components instead of using language based programming.

2 Phases of development from Spaghetti Code to Component Based Software Development

The development is continuous as well as gradual. A lot of phases can be distinquished, however, there were several side tracks, dead ends and success stories as well. The categories are defined voluntarily, according to which the following significant phases and trends are highlighted:

2.1 The "Spaghetti Code" paradigm

After the phase of assembly programming, which followed the era of machine level programming, the appearance of higher level programming languages was a redemption. These languages, however, had very few tools, their control structure was not easy to survey, solution of larger tasks could hardly be realised and managed.

2.2 The "Divide et Impera" paradigm

The principle is obvious and clear, the task is too big and too complex and it has to be splitted into smaller parts and after the realisation of the specific moduls, these small parts must be re-integrated. Naturally, to solve this problem, the appearance of such languages that make the subdivision of program into smaller parts, subroutines, functions, procedures, modules possible was crucial. This technique helped to reduce complexity, however, after a certain size the system itself becomes complex and hard to handle.

2.3 "The world is structured" paradigm

Structured rpogramming and structured approach have strengthened since the 70s'. The paper by Dijkstra was a mile stone, the appearance of Jackson's JSP as well as the appearance of PASCAL [1] by Wirth and ADA contributed to a great extent to the practical realisation.

Beyond programming, the structured approach of other fields of software engineering, analyses and planning, appeared as well. Michel Jackson [2], then later DeMarco [3], Stevens, Myers, Constantine [4] and Yourdon [5] are to be highlighted. Without completing the list, the methodologies by Gane, Sourcon and Warnier, and Orr must be mentioned as well.

2.4 "Think Object-oriented" paradigm

The paradigm that followed the structured one made programming at an even higher abstraction level possible by integrating data and the operations made on these data into an organic unit. In order to spread the OO paradigm, the

development of resources turned out to be necessary again, while we got to the nowadays used highly effective languages (C++, JAVA) from the first primitive programming languages.

In the Object Oriented Software Engineering, the first significant methodology was published by Booch [6]. A lot of methodologies turned up later, but the next important methodology, that was the most accepted in practice, was the Object-Oriented Analysis and Design noted by Coad-Yourdon, [7] [8] and completed by Jacobson [9] later. Apart from these the responsibility driven methodology by Rebeca Wirfs-Brock [10] was also introduced. Object Modelling Technic, which was published by Rainbow et Al., [11] was the first popular and widely used modelling-based method, which was the forerun of the later introduced and even nowadays the most successful Unified Modelling Technique [12]

2.5 Component based technique

The continuous development of paradigms gave birth to an old-new technology. The basic principle of component based sofware development is not at all new. The first proposal was presented at the famous NATO Software Engineering conference in 1968 in Garmisch (Germany). Object oriented technology gave a new impulse to the development of component elaboration and utilisation.

Object oriented principles can ensure component definition, component realisation in advance and component reuse the most efficiently. Components are more than simple objects, components are more complex units. Taking an example from electcronics, they are software integrated circuits (SW-IC) or even higher level units (PC-cards). Some say that the introduction of components meant not only a higher abstraction level but the birth of a new programming paradigm (component oriented or component based programming paradigm).

3 The methodology of component based software development

3.1 General features of the CBSD method

Component based software development made the realisation of large software systems by assembling previously developed components possible. The most important features of this methodology follow below:

- It decreases the total development cost of the system under development to a great extent.
- It decreases significantly the system development time.

- The quality and reliability of the total system improve thank to the better quality of previously developed and reused components.
- The follow up, maintenance, upgrade of software systems become cheaper, quicker and safer due to the simplier change of components.
- Priority in software development moves from software making to the integration of components and building from larger more complex units.
- Higher abstraction level makes the development of even larger and more complex systems possible.

3.2 The approach of CBSD method

According to Brown [13] CBSD approach has four major activities:

- Componet qualification
- Component adaptation
- Assembling components into systems
- System evolution

3.2.1 Component qualification

Component qualification is an activity or process to determine "the fitness for use" of the previously-developed components into the new system context. It is also a process to select the components if a market of these components exists (make-buy decision)

The component qualification has two phases:

- Discovery phase, to identify the properties of each component (which kind of components we need to solve the problem)
- Evaluation phase, for selecting from among a group of peer products (which the right components are from the given set to solve the problem)

The component qualification phase is a highly critical phase regarding the quality of the sytem under development.

3.2.2 Component adaptation

In several cases the already existing components do not fulfill entirely the demands of the new system that is why the adaptation of these components becomes necessary.

According to Valeto [14] the degree to which a component's internal structure is accessible differs 3 approaches to adaptation:

- White box approach (the source code of the component will significantly be modified)
- Grey box approach (the source code of the component is not modified but the component provides its own API (Application Programming Interface))
- Black box approach (only the binary code of the component is available and there is no even API))

3.2.3 Assembling components into systems

This activity describes the way how to integrate the preselected components into the new system, how to assembly them together. There are more architectural styles developing the system from components of the shelf such as Object Request Broker. ORB is a middleware technology that manages communication and data transfer between distributed objects in the system.

3.2.4 System evolution

Component Based Systems seem to be relatively easy to evolve, because to repair the failer means “just” the change of the defected component. Similarly to this, when an additional functionality is required, it could be realised through a new component, which is “just” added to the system.

This view is highly optimistic. In practice a “change or modification” means always a source of potential errors. The components must be tested in isolation as well as together with the other components of the system.

3.3 Realisations of Component Based Software Development

3.3.1 The “Microsoft Line”

Microsoft provides the COM technology (Component Object Model) for software components. With Windows 2000 a significant extension to COM, COM+ was released. COM+ could run in component farms managed with the Microsoft Transaction Server. The distributed version of COM is the DCOM, which is a technology for software components distributed across several networked computers. In 2002 .NET was released, which presents a platform-independent target for software development, it relies fully on software componentry and the component oriented programming paradigm. Microsoft stated clearly, that .NET will replace COM as a software component architecture.

3.3.2 The “Sun Line”

The success story of Java survives in component based development as well. The basic principle of Java originally included the development of networked,

distributed applications. Sun's reaction to CBSD was the Enterprise Java Beans (EJB). EJB architecture is a component based architecture for developing and deploying component objects. Applications written to the EJB specifications are scaleable, transactional and secure, they can be deployed on any platform that supports EJB. The EJB architecture will rely on standard component you developed with third—party components into a single application.

3.3.3 The "OMG Line"

The Object Management Group's (OMG) product is the Common Request Broker Architecture (CORBA), which is an infrastructure for handling components (objects). It provides the communication mechanisms between distributed objects. OMG's Interface Definition Language (IDL) describes the services of objects. The big advantage is that the Interface definition is independent from the programming language, but it maps to all of the popular programming languages via OMG standards (C, C++, Java, COBOL, Smalltalk, Ada, Lisp, Python, IDLscript).

4 Special Aspects of Component Based Software Development

The special aspect of the above examination is the application features of CBSD in Software Engineering higher education. In the above it was clearly defined that component based development has a lot of positive characteristics and its application in the software industry is continuously increasing. In parallel, education - following the demand of the industry – must introduce CBSD. However, the application of CBSD in education has special aspects:

- In the first place, CBSD can be well adapted in developing large systems, which ones – regarding their size - cannot be used in case of projects made by students.
- In order to gain practical knowledge and learn tricks and tweaks, it is practical to introduce case studies, in which the completion of subtasks can provide tasks to students.
- The application of CBSD deals with software development at a higher abstraction level, which can create such a false image in students without adequate programming pre-studies, that software development means only building from Lego bricks.
- During the application of Components Of The Shelf (COTS) technique the complexity, the exact description and number of components make the selection of the proper component quite difficult, thus the real life situation in education can hardly be presented.

- Learning the technologies that make the realisation of CBSD possible, (Microsoft-Line, Sun-Line, OMG-Line) requires a lot of prelearning and knowledge thus the offset-threshold is relatively high, which in several cases puts the students off from the technical adaptation.
- Education must be impartial, meaning that it must keep the same distance from each methodological trend, which means that it has to give the opportunity, if only optionally, to the presentation of each thecnology. Regarding the offset threshold of each thecnology, and the difficulties connected to them, this is not easy.
- The large resource demand of the technologies gives an obstacle in case of an individual project made by a student. It is quite laborious to ensure access for students in case of completing theses, Scientific student projects and other ones.

Conclusions

In summary, it can be said that component based software development is a modern and more and more widespread solution in the software industry. Responsing to the demand in the industry its place has to be ensured in higher education as well. The application of more and more complex and larger and larger systems, - like CBSD's as well – in education is not at all easy, and arises a lot of questions, the solution of which is the urgent tasks of the near future.

References

[1] Jensen, K., Wirth, N.: PASCAL – User Manual and Report
Springer Verlag, 1974

[2] Jackson, M.: Principles of Program Design
Academic Press, 1975

[3] DeMarco, T.: Structured Analysis and System Specification
Prentice-Hall, 1979

[4] Stevens, W. P., Myers, G. J., Constantine, L. L.: Structured Design
IBM Systems Journal, vol.13, no. 2, 1974

[5] Yourdon, E. N., Constantine, L. L.: Structured Design
Yourdon Press, New York, 1978

[6] Booch, G.: Object Oriented Design with Applications
The Benjamin/Cummings Publishing Company, Redwood City 1991

[7] Coad, P., Yourdon, E.: Object-Oriented Analysis
Yourdon Press, New York, 1991

[8] Coad, P., Yourdon E.: Object-Oriented Design
Yourdon Press, New York 1991

[9] Jacobson, I., Christerson, M., Jonsson, P., Övergaard, G.: Object-Oriented Software Engineering – A Use Case Driven Approach
Addison Wesley, 1992

[10] Wirfs-Brock, R., Wilkerson, B., Wiener, L.: Designing Object-Oriented Software
Prentice Hall PTR, Englewood Cliffs, New Jersey, 1990

[11] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: Object-Oriented Modelling And Design
Prentice Hall International Edition, 1991

[12] Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual
Addison Wesley, 1998

[13] Brown, A. W., Wallnau, K. C.: Engineering of Component-Based Systems
IEEE Computer Society Press, 1996.

[14] Valetto, G., Kaiser, G. E.: Enveloping Sophisticated Tools into Computer-Aided Software Engineering Environments
Proceedings of the 7th IEEE International Workshop on CASE, Toronto, Ontario, Canada, July 10-14, 1995, pp. 40-48.