Journal of AL-Qadisiyah for computer science and mathematics
Vol.8 No.1 Year 2016

Assessing Attributes to Software Resources for minimize the Development Life Cycle

Rafid Nabil Jaffar
Department of Computer Science College of Computer Science & IT Al-Qadissiya University
E-Mail: rafidnj7777@yahoo.com

Recived :8\2\2016 Revised : // Accepted :21\2\2016

Abstract:

The Component-Based Software Engineering (CBSE), approach emphasize on acquisition and integration of components to accomplish complex and large-scale software solutions. The benefits from using a CBSE approach include, system quality improvement, shorter time-to-market, and improved management of complexity of software. However, the focus of development move to issues like selection, integration, evaluation and evolution of components in the system. Underestimating the technical risks associated with selection, evaluation, and integration of software components can result in long schedule delays and high development/maintenance cost.

This paper introduces basic concepts of CBSE and Commercial-Off-The-Shelf, COTS, components. Driving factors for the use of COTS components are presented together with potential benefits and key issues to consider in order to successfully adapt to a CBSE approach. The intent is also to point out possible risks that are typically not present when developing traditional, and not use CBSE for software systems. Specifically the basic CBSE issues of system reliability, development process, and real-time system development are presented.

Keywords: Reuse Software, COTs, Component Based Software Engineering, Reliability of Reuse, Real-Time Systems .

1. Introduction:

Software developers have long dreamed about system development using pre-prepared components in a package style. The vision is that a new system should be built by assembling components and that the new system should be functional with little effort since the components are already developed, tested, and matured by long execution in many different contexts. Over the last decades we have seen the focus of software development shift from individual systems to assembly of large numbers of components into systems or families of systems. Traditional development is changed to focusing on selection, evaluation, integration, and evolution of reusable software components. Today some systems include such complex components that developing them from scratch would be impossible if a profit is required.

However, the vision of package style assembly of software systems is not without problems. Underestimating the technical risks associated with selection, evaluation, and integration of software components can result in long schedule delays and high development/maintenance cost [10]. This paper presents a survey of relevant papers to the field of CBSE and development using COTS components. The first part introduces basic concepts to CBSE and COTS and general advantages and disadvantages of applying a CBSE approach to software development. The second part introduces various issues like reliability and development process considerations when using COTS and a CBSE approach. The third part summarizes the paper and present some recommendations presented by [3].

1.1 Reuse Software

Today the trend in Computer-Based products, such as cars and mobile phones, is shorter and shorter lifecycles. As a consequence, time spent on development of new products or new versions of a product must be reduced. One solution to this emerging problem is to reuse software design and solutions in new versions of systems and products. Besides shortening development time, properly handled reuse will also improve the reliability since code is executed for longer time and in different contexts [16]. However, reuse is not trivial and puts strong demands on development methods in order to be successful. When applying reuse in development of real-time systems, methods gets even more complex since both functional behavior and temporal behavior must be considered.

1.2 Why Reuse Software?

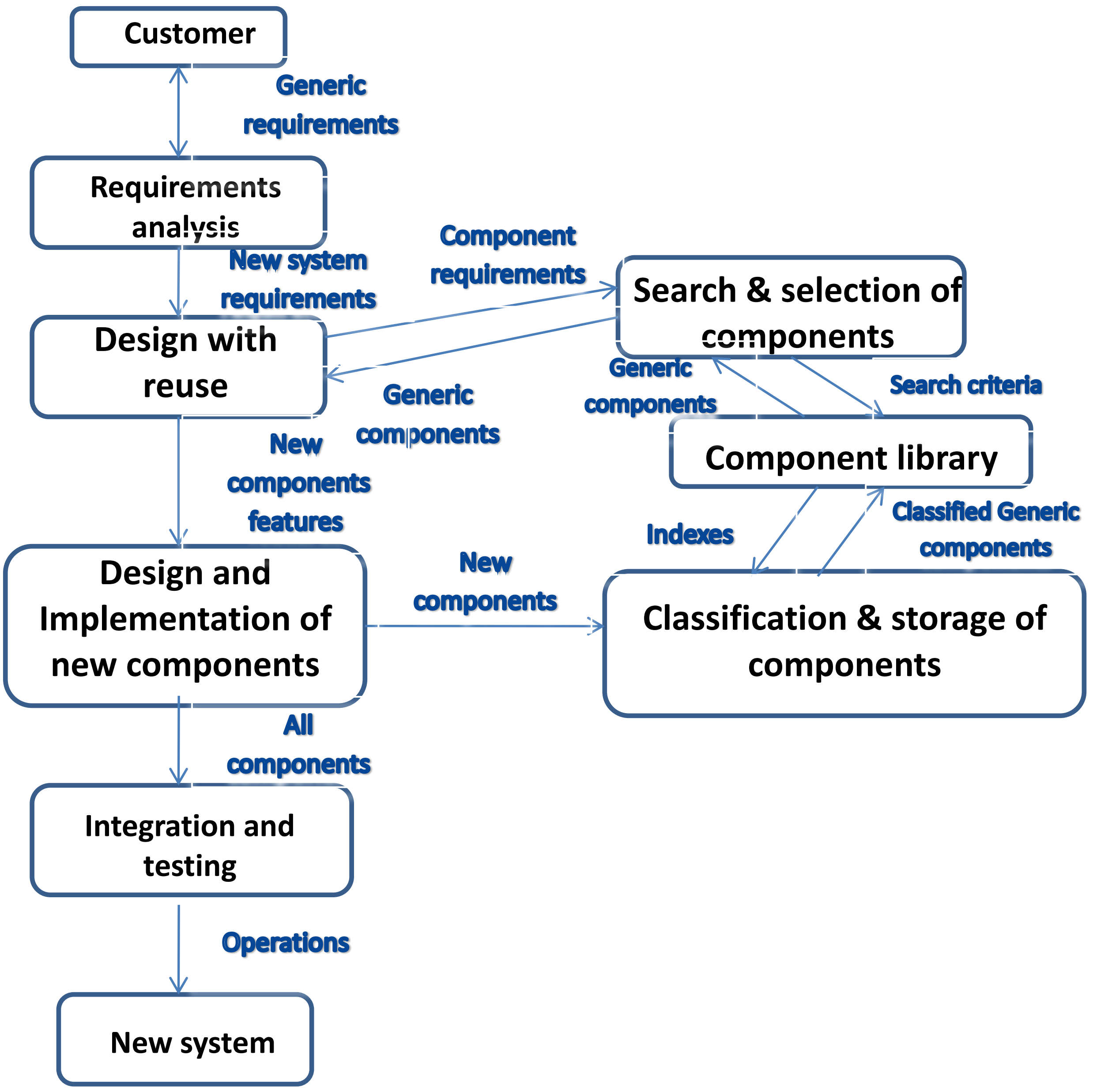
A good software reuse process facilitates the increase of productivity, quality, and reliability, and the decrease of costs and implementation time. An initial investment is required to start a software reuse process, but that investment pays for itself in a few reuses. In short, the development of a reuse process and repository produces a base of knowledge that improves in quality after every reuse, minimizing the amount of development work required for future projects and ultimately reducing the risk of new projects that are based on repository knowledge.

1.3 Systematic software reuse:

One way of approaching the issue of reuse is to develop systems by a Component-Based Software Engineering, CBSE, paradigm: assembling software systems from components by package style. Mainly three improvements in computer and software-based systems can be expected by a CBSE approach to system development.

- Improvement of system quality
- Achievement of shorter time-to-market
- Improved management of increased complexity of software

In order to achieve these improvements, the software industry has been moving very rapidly in defining standards and developing component technologies. Systematic software reuse and the reuse of components influence almost the whole software engineering process (independent of what a component is) [17]. Software process models were developed to provide guidance in the creation of high-quality software systems by teams at predictable costs. The original models were based on the misconception that systems are built from scratch according to stable requirements. Software process models have been adapted since based on experience, and several changes and improvements have been suggested since the classic waterfall model like sketch below (Figure-1), with increasing reuse of software, new models for software engineering are emerging.



Fig(1) reuse in waterfall model

New models are based on systematic reuse of well-defined components that have been developed in various projects [17]. Developing software with reuse requires planning for reuse, developing for reuse and with reuse, and providing documentation for reuse. The priority of documentation in software projects has traditionally been low [17]. However, proper documentation is a necessity for the systematic reuse of components. If we continue to neglect documentation we will not be able to increase productivity through the reuse of components. Detailed information about components is indispensable. Although the track record for systematic software reuse has been rather spotty historically, several key trends good technology for software reuse in the future:

- Component- and framework-based middleware technologies, such as CORBA, J2EE, and .NET, have become main stream.
- An increasing number of developers of projects over the past decade have successfully adopted OO design

techniques, such as UML and patterns, and OO programming languages, such as C++, Java, and C#. These trends are particularly evident in markets, such as electronic commerce and data networking, where reducing development cycle time is crucial to business success. Although there is no magic methodology or process that's guaranteed to foster systematic reuse, I have personally seen the recommendations below applied successfully numerous times over the past decade on many projects at many companies around the world. I give the example for the Component-Based Development Model (CBD) like sketch below (Figure-2) that incorporates many of the characteristics of the spiral model. It is evolutionary in nature [NIE92], demanding an iterative approach to the creation of software. However, the Component-Based Development model composes applications from prepackaged software components (called *classes*). The engineering activity begins with the identification of candidate classes. This is accomplished by examining the data to be manipulated by the application and the algorithms that will be applied to accomplish the manipulation. Corresponding data and algorithms are packaged into a class.

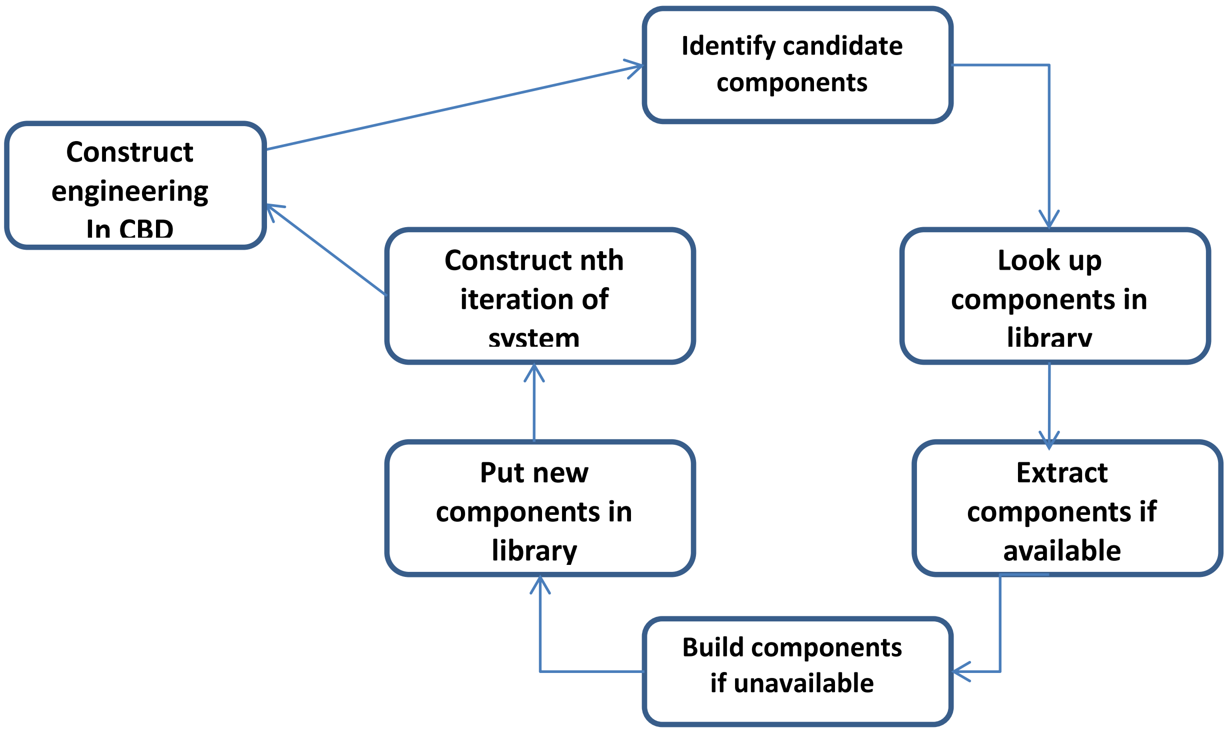

Identify candidate components
Construct engineering In CBD
Construct nth iteration of system
Look up components in library
Put new components in library
Extract components if available
Build components if unavailable

Fig (2) A reuse in CBD

1.4 Real-Time Components:

Real-time systems are systems in which the correctness of the system depends not only on the result of the computations it performs but also on timing behavior [4]. Many real-time systems are also embedded systems that interact with external devices in a product, which if malfunctioning, often can cause more damage than a desktop software system. Therefore, real-time systems must usually meet stringent specifications for safety, reliability, limited hardware capacity etc. The increased complexity of real-time systems leads to increasing demands with respect to high-level design, early error detection, productivity, integration, verification and maintenance. Applying Component-Based Software Engineering (CBSE) methodology in the development of real-time systems, is more complex and more expensive than designing non real-time components [2,5]. This complexity arises from several aspects of real-time systems not relevant in non-real-time systems. In real-time systems, components must collaborate to meet timing constraints. Furthermore, in order to keep production costs down, embedded systems resources are usually scarce, but they must still perform within tight deadlines. They must also often run continuously for long periods of time without maintenance. An interesting observation about efficient reuse of real-time components [15] is that, as a rule of thumb, the overhead cost of developing a reusable component, including design plus documentation, is recovered after the *fifth* reuse.

1.5 COTS (Commercial Off-The-Shelf) for Software reuse:

Reusing components made for earlier products as an approach to new system development is a promising way of achieving the mentioned development and system improvements. There is also the possibility to buy software components from component vendors, so called Commercial-Off-The-Shelf, COTS, components that consider horizontal reuse. The use of Commercial-Off-The-Shelf or third-party application within a larger system, such as an email package or a word processing program. The software components is increasing in today's development of new systems. Shorter system life cycles and decreased development budgets make it so. Using COTS Components can be one way of reducing development time and be competitive by getting products to the market fast and inexpensively.

COTS components can also provide an increased reliability compared to custom-made components since they are refined by substantial field-testing. Certain types of components are rarely developed only for one intended system anymore. Development of a database management system or an operating system as a part of a larger project is almost unthinkable for many applications today. A summary of the advantages that can be gained by developing a system using COTS components:

- Functionality is instantly accessible to the developer.
- Components may be less costly than those developed in-house.
- The component vendor may be an expert in the particular area of the component functionality.

Although, using COTS Components can save valuable development time, insight in the COTS component functionality and properties must be evaluated for its intended use. In order to integrate a COTS component in a system, the developers must consider relevant properties of the component like operational limitations, temporal behavior, preconditions, robustness and many forms of underlying assumptions on the intended environment. To determine its properties, extensive testing of the component may be necessary [12].

2- COTS (Commercial-Off-The-Shelf) in System Development:

The question whether to buy or develop a component can require an extensive examination in the case of each component in order to determine buy or

build. In order to make decisions, several aspects of COTS usage must be understood and evaluated. Moreover, the general benefits and challenges should be understood.

2.1 COTS Driving Forces:

The Component-Based Software Engineering approach emphasize on acquisition and integration of components to accomplish complex and large-scale software solutions. Components can be either developed in-house or off-the-shelf. The main driving factors for using COTS components rather than developing the whole system in-house are presented in [6] and includes:

1. Industrial competition for delivering more reliable systems in shorter time frames.
2. A demand for larger and more complex software solutions, which often
cannot be effectively implemented in a timely manner by a single software development organization.
3. Increase in availability of reusable COTS components.
4. Increased degree of standard compliance among COTS software products that enables reduction of product integration time.
5. Increasing research in better software component "packaging" techniques and
approaches.
6. Increasing recognition that software reuse is one of the most important means to achieve better software solutions with minimum development cost.

2.2 COTS Challenges:

There are several challenges when using COTS components instead of developing in-house. Assessing functionality, integration, operational profile, quality for intended use, performance and other properties may require substantial effort. Developers must have a good understanding of the COTS component relevant properties. General guidelines on which properties
are relevant are hard to present and
depend on the system to be developed. A summary of disadvantages are presented in [13] and includes:

- Often, only a brief description of its functionality is provided with a COTS component.
- A component, often, carries no guarantee of adequate testing for the intended environment.
- There is no, or only a limited description of the quality of the component and the
quality must be assessed in relation to its intended use.
- The developer, typically, does not have access to the source code of the component.

To make the decision to buy or to build is not easy, knowing all the disadvantages. COTS components are typically "black boxes" without their source code or other means of introspection available. Developers must assess a number of properties of the wanted COTS component to integrate it properly with a system under development. Examples of relevant properties are functionality, fault behavior, temporal behavior, preconditions, robustness and performance. To determine its properties, extensive testing of the component may be necessary. There are various approaches to this kind of testing, e.g. random, "black-box" and "white-box" test generators.

2.3 Reliability Issues with COTS:

The issue of system reliability differs when using COTS components compared to traditional development. The key in integrating COTS components is to understand the components properties like its intended environment and the assumptions under which it was developed. Any discrepancies must be handled in order to perform a successful integration. There are several risks involved when using COTS products. General issues to consider when using COTS are presented in many papers [3,9,12] and a short summary is presented here:

- Assure COTS components are applied within their intended profile.
- Understand and document how the COTS components behave in a fault situation.
- Use guidelines and tools to deal with supplier changes and upgrades of the COTS component.
- Determine if future releases of the COTS component are backward compatible.
- Investigate what development procedure has been used and if it complies with any reliability standards.

The model was presents for determining the reliability of components for the software[8]. To acquire confidence in a component it must be supplied with a contract and be tested with a certain input. A contract specifies the functionality and the run-time conditions for which the component has been designed, i.e. assumptions about inputs, outputs and environment. If the component supplier provides such a contract, it can be used to calculate the probabilities of the occurrence of errors. Evidence based on the component contracts and the experience accumulated must be obtained. The environment must be considered when components are integrated in new systems; the input domain may differ considerably from the input domain for which it was tested. Confidence in a component's reliability is only warranted when the component is used in the environment for which it is intended.

2.4 Development Process Issues:

The principal danger of using COTS components in safety critical applications lies in the discontinuity it creates in the understanding of the system as a whole [7].

The designer of the component may not have full knowledge of the application or systems where their component is to be used. Moreover, the application developer may not fully understand the component's intended use and underlying assumptions. When developing an application and including COTS components, it may be beneficial to set up teams consisting of both application developers and the COTS developers to get a common understanding of the COTS behavior and intended use. Such teams facilitate better relationships between the user and the supplier, which makes investigation of the supplier's development process easier. Any COTS components intended to be used in safety-critical applications should be thoroughly tested in its intended environment. If unreliable or unnecessary parts are found, it can be possible to wrap those parts with protective code. One way of performing tests of COTS components is to use fault injection [3] that can reveal the consequences of failures in COTS components on the rest of the system. When the component vendor wants to express certain capabilities of the component, it is important to set the context in which each reliability argument is described. For instance, if a vendor claims that his component provides on average 10^4 hours continuous fault free operation then it all sounds fine, but what is the claim based on? By using Goal Structuring Notation (GSN) [14], it is possible to structure and present the required context information. GSN uses a graphical notation and the example above would reveal that the argument is based on certain assumptions and operational data from only 50% of the tests covered. Using this notation, it is easier for the safe designer to evaluate the component with less chance of misunderstanding.

2.5 Real-Time Systems and Components:

The analyze of the possibility to apply CBSE to the development of hard real-time systems[2]. Developing real-time systems present a different setting compared to many large-scale business systems. Many real-time systems are also embedded systems that will be delivered together with a product in which they are

embedded. This often causes hardware resources to be very sparse to keep the product cost low. Hence, real-time system developers must often ensure that they are using the target hardware resources very efficiently. Common
component technologies, such as JavaBeans, CORBA and COM, are rarely used, due to their requirements on expensive hardware and their unpredictable timing characteristics. However, a model for hard real-time systems cannot support flexibility to the same extent as these common component technologies. Compared to a regular component, a real-time component must include the specification of timing requirements. Timing behavior is dependent on the target architecture and the memory organization. This leads to that a component's WCET can vary.

3- Conclusions:

System development with a CBSE approach can be complex process, as several issues must be considered before deciding to buy or to develop a wanted component. Generally, it is often better to buy general-purpose components, e.g. operating systems, databases and user interface components and to develop domain specific components. Many different aspects must be considered before choosing an existing component over an internal. The development of proprietary components takes resources, requires maintenance and support, but gives better freedom as to the exact specification. Buying a component on the other hand, saves development time, but may require substantial effort in integration and perhaps negotiation of the requirements. Before making decisions on buy/build issues for CBSE development, the following questions and thoughts should be considered:

• The functionality provided by a COTS component may not remain the same with newer versions of the component. This may force creation of encapsulate for the component, which provide or prevent functionality. If the support from the component vendor is inadequate, this could be a serious issue. If unwanted functionality is removed, the price may be paid unnecessarily.

• Even if the source code is available from the component vendor, can our organization maintain it if something goes wrong?

• If an external component is customized for a product, it makes the product strongly dependent on the component vendor. The vendor can then set his own price for continued support of the component. There are many issues surrounding CBSE to be addressed before making decisions on how to design a system with components. The following recommendations to the component integrator [3,11]:

• Make a thorough evaluation of the component suppliers. Are they suitable as suppliers? Do they have good quality products and support? Check their financial position for economic stability.

• Ensure that the legal agreement with
the supplier is comprehensive. This may save time and efforts if the supplier goes out of business or if they refuse support of their component.

• Create good and long term relations with the supplier for better cooperation.

• Limit the number of partners and suppliers. Too many will increase the costs and the dependencies.

• Buy "big" components where the profit is greatest. The management of too many small components can consume the profit.

• Adjust the development process to a Component-Based process.

• Have key persons assigned to supervise the component market, monitoring new components and trends.

• Try to gain access to the source code. Through special agreements with the vendors.

- Test the components in the target environment.
- Assure COTS components are applied within their intended profile.
- Understand and document how the COTS components behave in a faulty situation.
- Use guidelines and tools to deal with supplier changes and upgrades of the COTS component.
- Determine if future releases of the COTS component are backward compatible.
- Investigate what development procedure has been used and if it complies with any reliability standards.

These recommendations do not provide a complete solution to all the problems that may occur, but they indicate that developing for and with components must be performed carefully.

4- <u>References</u>:

[1] Vigder, M., Dean, J., *Building Maintainable COTS Based Systems*, Proceedings, International Conference on Software Maintenance, 1998.

[2] Isovic, D., Lindgren, M., Crnkovic, I., *System Development with Real-Time Components*, In Proc. of ECOOP2000 Workshop 22 - Pervasive Component-based systems, June 2000.

[3] Besnard, J. F., Keene, S. J., and Voas, J., *Assuring COTS Products for Reliability and Safety Critical Systems*, Reliability and Maintainability Symposium, Proceedings IEEE Computer Society, 1999.

[4] Stankovic, J. and Ramamritham, K., *Tutorial on Hard Real-Time Systems*, IEEE Computer Society Press, 1998

[5] Douglas, B.P., *Real-Time UML - Developing efficient objects for embedded systems,*Addison Wesley Longman, Inc, 1998.

[6] Tran, V., Liu, D., *Component-based Systems Development: Challenges and Lessons Learned*, Software Technology and Engineering Practice, Proceedings., Eighth IEEE International Workshop on, 1997.

[7] Dawkins S. and Kelly T., *Supporting the Use of COTS in Safety Critical Applications*, In *Proceedings of IEEE Colloquium on COTS and Safety Critical Systems*, 1997.

[8] Thane H. *Monitoring, Testing and Debugging of Distributed Real-Time Systems*, Doctoral Thesis Royal Institute of Technology, KTH, Mechatronics Laboratory, TRITA-MMK 2000:16, Sweden 2000.

[9] Profeta, J.,Andrianos, N., Yu, B., *Safety-Critical Systems Built with COTS*, IEEE Computer, Volume: 29 Issue: 11 , Nov. 1996

[10] Ning, J., *Component-Based Software Engineering (CBSE),*
Assessment of Software Tools and Technologies, Proceedings Fifth International Symposium on, IEEE,1997

[11] Josefsson, M., *Program varu komponenter i praktiken -att köpa tid och presteramer*, paper V040078, Sveriges Verkstads industrier, 1-1-1999.

[12] Voas J., *COTS Software: the Economical Choice?*, IEEE Software, volume 15, issue 2,1998.

[13] Korel B., *Black-Box Understanding of COTS Components*, In Proceedings of 7th international workshop on program comprehension, 1999.

[14] Wilson S. P., Kelly T. P., and McDermid J. A., *Safety Case Development: Current Practice, Future Prospects*, In Proceedings of 12th Annual CSR Wokshop, Springer-Verlag, 1995.

[15] Mrva, M. *Reuse Factors in Embedded Systems Design*. High-Level Design Techniques Dept. at Siemens AG, Munich, Germany, 1997.

[16] N. E. Fenton, S. L. Pfleeger, *Software Metrics: A rigorous & practical approach*, International Thomson Computer Press, ISBN 0-534-95600-9, 1996.

[17] Sametinger, Software Engineering with Reusable Components, Springer-Verlag, ISBN 3-540-62695-6, 1997.

تقييم خصـائـص موارد البرامجيــات لتقليـص دورة حياة بنــاء النظــام

رافد نبيل جعفر
جامعة القادسية / كلية علوم الحاسوب وتكنولوجيا المعلومات / قسم علوم الحاسوب
rafidnj7777@yahoo.com

المستخلص :

في هندسة البرمجيات المستندة إلى المكونات (CBSE)، نهج يركز على اكتساب وتكامل بين المكونات لإنجاز الحلول البرمجية المعقدة وعلى نطاق واسع، وتشمل الفوائد من استخدام المكونات المبنية مسبقاً الى تحسين نوعية النظام، تقليص وقت بناء المنتج وأطلاقه للأسواق، وتحسين إدارة تعقيد البرمجيات. ومع ذلك، فإن التركيز على البناء ينتقل إلى قضايا مثل الاختيار (الانتقائية) والتكامل والتقييم وتطور المكونات في النظام. يمكن التقليل من المخاطر التقنية المرتبطة بالاختيار (الانتقائية) ، والتقييم، والتكامل بين مكونات البرامجيات والذي يقود إلى التأخير في الجدول الزمني للبناء والكلفة العالية للبناء والادامة.

ويقدم هذا البحث المفاهيم الأساسية لـ هندسة البرمجيات المستندة إلى المكونات (CBSE) والبرامجيات التجارية الجاهزة (COTS). ان العوامل التي تدفع لاستخدام البرامجيات التجارية الجاهزة (COTS) بالإضافة الى الفوائد المتوقعة والامور الأساسية التي تؤخذ بنظر الاعتبار من أجل أنجاح عملية التعديل على المكونات التي تم بنائها مسبقاً. والقصد من ذلك هو أيضا الاشارة إلى المخاطر المحتملة التي عادة ما تكون غير موجودة عند البناء بالطرق التقليدية، حيث ومن القضايا الاساسية لـ (هندسة البرمجيات المستندة إلى المكونات) مفهوم موثوقية النظام، وعمليات البناء، وتطوير وبناء أنظمة الزمن الحقيقي .