IJCSMS International Journal of Computer Science and Management Studies, Vol. 11, Issue 03, Oct 2011
ISSN (Online): 2231-5268
www.ijcsms.com

Component Based Effort Estimation During Software Development: Problematic View

VINIT KUMAR
Research Scholar, MBU Solan (India)
lohanvinit@gmail.com

Abstract
Component-based software development (CBD) is an emerging discipline that promises to take software engineering into a new era. Building on the achievements of object-oriented software construction, CBD aims to deliver software engineering from a cottage industry into an industrial age for Information Technology, wherein software can be assembled from components, in the manner that hardware systems are currently constructed from kits of parts. Component-based development (CBD) is a branch of software engineering that emphasizes the separation of concerns in respect of the wide-ranging functionality available throughout a given software system. This practice aims to bring about an equally wide-ranging degree of benefits in both the short-term and the long-term for the software itself and for organizations that sponsor such software. Software engineers regard components as part of the starting platform for service-orientation. Components play this role, for example, in Web services, and more recently, in service-oriented architectures (SOA), whereby a component is converted by the Web service into a *service* and subsequently inherits further characteristics beyond that of an ordinary component. Components can produce or consume events and can be used for event driven architectures (EDA).
Keywords: *Component based software development, effort estimation metrics parameters, reusability, and maintainability.*

Introduction

Component based software development [2, 3] is a dream of the software industries, where programmers would become merely assembly workers and development process of a new software system would be similar to assembling. And it is demand of today software market because today software project is becoming more and more complex and is hard to manage and control.

Here we will introduce new paradigm for software development as well as provided metric for effort estimation that will improve the complexity of component, dependency and composite of component based software development. With the help of metrics, a bottom-up measuring process from component to the system can full fill evolution for component oriented software development complexity. The purpose of metrics is characterized with the simplicity, reusability, portability, maintainability, reliability etc.
The idea behind component based software development approach is, develop software system by selecting appropriate off-the-shelf component and then to assemble them a well defined software architecture. It is new approach in software engineering community. The purpose of component based software engineering is to develop large system that incorporate previously developed or existing component, thus cutting down an development time and cost. It can also reduce maintenance associated with the upgrading of large system.

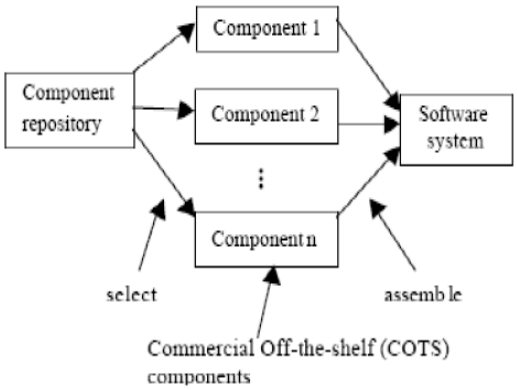
Historical Background

In the 1960s, programmers built scientific subroutine libraries that were reusable in a broad array of engineering and scientific applications. Though these subroutine libraries reused well-defined algorithms in an effective manner, they had a limited domain of application. Commercial sites routinely created application programs from reusable modules written in Assembler, COBOL, PL/1 and other second- and third-generation languages using both System and user application libraries.

As of 2010, modern reusable components encapsulate both data structures and the algorithms that are applied to the data structures. It builds on prior theories of software objects, software architectures, software frameworks and software design patterns, and the extensive theory of object-oriented programming and the object oriented design of all these. It claims that software components, like the idea of hardware components, used for example in telecommunications, can ultimately be made interchangeable and reliable. On the other hand, it is argued that it is a mistake to focus on independent components rather than the framework (without which they would not exist)

COMPONENT

In defining a software component, this definition can be quoted: "A component is a software element that conforms to a software model and can be independently deployed and composed without modification according to a composition standard" [7]. CBSE is about creating a software package in such a manner as to be able to easily reuse its constituent components in other similar or dissimilar applications. It includes writing high level code that glues together pieces of pre-built functionalities or software building blocks called components. Component is one of the parts of the system that make up a system. It may be hardware, software or firmware and may be sub divided into other components



'Component based software development'

A. Independent Software Development

Large software systems are necessarily assembled from components developed by different people. To facilitate independent development, it is essential to decouple developers and users of components through abstract and implementation-neutral interface specifications of behavior for components.

B. Reusability

While some parts of a large system will necessarily be special-purpose software, it is essential to design and assemble pre-existing components (within or across domains) in developing new components.

C. Software quality

A component or system needs to be shown to have desired behavior, either through logical reasoning, tracing, and/or testing. The quality assurance approach must be modular to be scalable.

D. Maintainability

A software system should be understandable, and easy to evolve [5],[6].

Research goal

Component-based Software development is a promising way to improve quality, time to market and handle the increasing complexity of software management.
It has been proven that software complexity is one of the major contributing factors to the cost of developing and maintaining software. Meanwhile, effort estimation is one of critical factor that directly affect the reusability, portability, reliability and maintainability.
In component based software development the architecture complexity is mainly attributable to the dependencies between component, such as procedure call, message passing and conversation protocol. Here we will introduce component based metrics that will directly affect on the interface among component and component interface is the key factor of component complexity.
In the context of software effort estimation [1], system sizes the taken as a main driver of the system development effort. But other structure

design properties, such as coupling, cohesion and complexity have been suggested as additional factor. Here, using effort data from component oriented development project [2,3], we empirical investigate the relationship between component size and development effort for a component and what additional impact structural properties such as connectivity, component interfacing have an effort. This paper proposes a practical, repeatable and accurate analysis procedure to investigate relationship between component properties and development effort.

Objective of the Proposed Research

System reliability is increased, reused components, which have been exercised in working systems, should be more reliable than new components. These components have been tested in operational systems and have therefore been tested to realistic operating conditions.
Overall process risk is reduced if a component already exists; there is less uncertainty in the costs of reusing that component than in the costs of development. This is an important factor for project management as it reduces the uncertainties in project cost estimation. This is particularly true when relatively large components such as sub-systems are reused.
Software development time can be reduced. Bringing a system to market as early as possible when it is mostly wanted is often more important than overall development costs. Reusing components speeds up system production because both development and validation time should be reduced.
Effective use can be made of specialist. Instead of application specialists doing the same boring work on different projects, these specialists can develop interesting new reusable components which encapsulate their knowledge.

Purposed Work

Here we will measure the effort of software project that is to develop based on component technology, such as COM/DCOM. COM/DCOM is general architecture for component software. It will define how component and their client interact directly and dynamically. DCOM is a protocol, that enables software components to communicate directly over network. These are designed for use across multiple network transports, including internet protocol such as HTTP.
COM AND DCOM HAVE PROVIDED a foundation for building component-based applications. Although they were initially available only on Windows platforms, the ongoing porting efforts to all major versions of Unix and mainframes (11) might turn COM/DCOM into a major cross-platform integration tool. The next generation of COM, called COM+, aims at simplifying the construction of COM applications by providing support in languages and tools and by providing a set of essential object services.

Main features

COM/DCOM's main features—*transparency*, *extensibility*, *and indirection*, *versioning*, and *server-lifetime management*—make it an attractive platform for research and development of distributed systems and applications.

Scope for future

The component oriented software project is implemented based on Microsoft technology such as COM/DCOM. Here we have been designed component oriented metrics that are used to determine effort of component oriented software. These metrics are designed in such a way that it will reduces more than 64 percentage effort of software as compared to meta mata metrics that are used to determine effort of traditional software development..

REFERENCES

1. A. M. Zaremski and J. M. Wing, "Specification matching of software components", ACM Transactions on Software Engineering & Methodology, 6(4):333-369, October 1997.
2. X.Cai, M.R. Lyu, K. Wong, Component-Based Soft-ware Engineering: Technologies, Development Frameworks, and Quality Assurance Schemes, Pro-ceedings APSEC 2000, Seventh Asia-Pacific Software Engineering Conference, Singapore, December 2000, pp 372-379.

3. 3.X.Cai, M.R. Lyu, K. Wong, Component-Based Soft-ware Engineering: Technologies, Development Frameworks, and Quality Assurance Schemes, Pro-ceedings APSEC 2000, Seventh Asia-Pacific Software Engineering Conference, Singapore, December 2000, pp 372-379
4. D. Box, Essential COM, Addison-Wesley, Reading, Mass., 1998.
5. G.C. Hunt and M.L. Scott, A Guided Tour of the Coign Automatic Distributed Partitioning System, Tech. Report MSR-TR-98-32,Microsoft Research, Redmond, Wash., 1998.
6. Y.M. Wang and W.J. Lee, "COMERA: COM Extensible Remoting Architecture," Proc. COOTS '98: Fourth USENIX Conf. Object-Oriented Technologies and Systems, Usenix, Berkeley, Calif., 1998,pp. 79–88; http://www.research.microsoft.com/~ymwang/papers/COOTS98CR.htm.
7. George T. Heineman and William T. Councill, "Component-Based Software Engineering Putting the Pieces Together", Addison-Wesley, Boston, MA ,880, June 2001.
8. "The Component Object Model Specification," Microsoft Corp., Redmond, Wash.,1995; http://www.microsoft.com/com/comdocs.htm.
9. "COM+," Microsoft Corp., 1998; http://www.microsoft.com/com/complus.htm.
10. 10 . "DCE 1.1: Remote Procedure Call Specification," The Open Group, Cambridge, Mass., 1997; http://www.rdg.opengroup.org/public/pubs/catalog/c706.htm.
11. N. Brown and C. Kindel, "Distributed Component Object Model Protocol—DCOM/1.0," Microsoft Corp., 1998; http://www.microsoft.com/com/.
12. Don Box, Essential COM, Addison Wesley, 1998.
13. "Microsoft Transaction Server," Microsoft Corp., 1998; http://www.microsoft.com/com/mts.htm.
14. "Millennium: Self-Tuning, Self-Configuring Distributed Systems," Microsoft Research, 1998; http://research.microsoft.com/sn/Millennium