J. Appl. Environ. Biol. Sci., 4(8S)13-17, 2014

ISSN: 2090-4274
Journal of Applied Environmental and Biological Sciences
www.textroad.com

An Efficient System For Generating Reports Of Cots Used In Component Based Software Engineering

Syed Hussain Abid[1], Shamila Nasreen[1], and Samina Khalid[2]

[1]MS Scholar, Department of Software Engineering, University of Engineering & Technology Taxila, Pakistan,
[2]Ph.D Scholar, Department of Software Engineering, Bahria University Islamabad, Pakistan

Received: September 1, 2014
Accepted: November 13, 2014

ABSTRACT

The world is advancing towards 4th generation techniques of software engineering (SE) and new paradigms are being introduced in every area of SE to outfit new demands of the progressive world. The primary focus of Project Management is time and cost. We are trying our level best to make processes fast and more responsive in order to overcome the time and budget constraints. In the area of software engineering, development of a new system is a comparatively slow process due to which most software projects lag behind the schedule and infringe the deadlines which in return have direct impact on expenditure. To overcome this problem Component Based Software Engineering (CBSE) was introduced which opened up into a new dimension in software development process. Using commercially off the shelf components (COTS) have proved to be an immense assistance to developers reducing a great amount of development time and cost. But their lies a problem in using COTS or a prewritten module or code i.e. we have to customize it as per our requirements. This customization requires complete understanding of the module as a part or the component as whole.

To understand a prewritten code is never an easy task. We propose an efficient and effective system for report generation of input software components that will be a great assistance for programmers who frequently need to customize codes or COTS according to organization's need.

KEYWORDS: SE, COTS, 4th GENERATION SOFTWARE ENGINEERING, CBSE.

INTRODUCTION

World is making progress by leaps and bounds and the evolution of technology is paving new paths in every field of knowledge. Like all other branches of Knowledge, Software Engineering is also making its contribution in world's advancement. It will be in no way wrong to say that Software Engineering is providing a mandatory support to world's progress.

Software engineering is a process that starts from an idea and ends up into a grand system. Like all other sciences, Software Engineering has some paradigms and heuristics which makes the foundation of Software development process. With the passage of time more efforts are being used in finding ways to make software development process faster and easier. At the same time the production of more supple, manageable and less error prone software systems are also serving as primary milestone to be achieved and focused in every software development life cycle.

Ever since the birth of Software Engineering, there have always been many approaches for software development and they all evolved with time and became more generic and applicable. They cover waterfall, evolutionary and many other software development methodologies. Many eras have passed and now world has entered in 4th generation techniques of Software Engineering. This opened up a new way of development i.e. component based development which includes using COTS, DLLs, Services and many more.

The question arises that why custom development has evolved into component based development of new system and when to make the decision between make and buy? What were the issues with custom development which are being addressed in component based development and what are the issues and complexities in component based development? Does the constraints on component based development strong enough to compliment the custom development?

Not only these but also other queries will be addressed in this paper and we also propose a system that will discourse one most concrete issue in component based development process. The propositions start with the fact that when the decision has been made to use any component then the obstacle arises to customize that component which require a thorough understanding of the plugins and internal architecture of the system. To understand someone else's program means to be in that state when he was coding and also to understand the approach with which he was coding. This is impossible at least without supernatural powers that we lack in this fragile world.

The solution is that there must be a tool that can generate a report about the component by doing a thorough examination of the component. The report will not only be a source of understanding about the component but will also help engineer who wants to customize that component.

COMPARISON BETWEEN CUSTOM DEVELOPMENT AND COMPONENT DRIVEN DEVELOPMENT

2.1. Cost and Budget Comparison:

A software development process starts with the requirements gathering and elicitation then after the design and implementation, exhaustive testing is done and then the product is deployed and maintenance phase starts and this cycle

Corresponding Author: Syed Hussain Abid, Department of Software Engineering, University of Engineering & Technology Taxila, Pakistan, hussainabid99@yahoo.com

continues for every new requirement and fault. This is the paradigm of the most custom software development and the most generic one too. While discussing the cost and budgetary of a software life cycle we can use a simple Bar chart as in Fig.1

These results were obtained after careful analysis of different Projects at Islamabad. The results show a clear distinction between the costs of custom development and component based development, this is due to the fact that, using a well-managed and trustworthy component can reduce the cost of maintenance. As long term maintenance cost can be reduced by using COTS [1] , but integration cost is the one, most affected by the software system architecture.

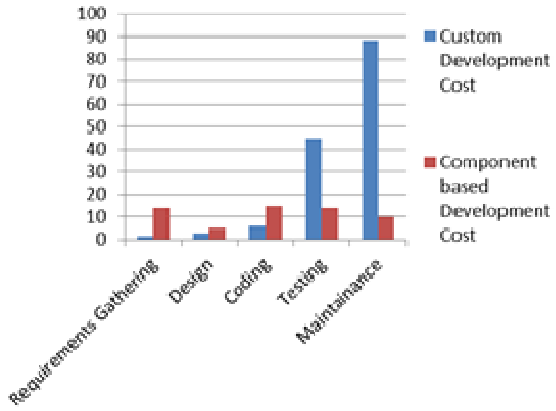

Figure 1: Cost Comparison between CBSE and Custom Development.

Integration and testing cost of component based development is high because component must be tested properly for maximum scenarios. Requirement and coding cost in component based development is also high than custom, because requirements also involve the cost of choosing the best and suitable component and coding involves the cost of customization. As far as the customization is concerned we must know the internal details of the component which we are generating by our proposed system. This will further reduce the cost of coding in component based development by reducing the learning time of the component.

2.2. Schedule and Time Comparison:

Reduced Cycle time and improved productivity with fewer people means lower cost [2]. The foremost objective of today's progressive world is time. In software development time management is a primary concern that prevails from requirements gathering till deployment. Time and schedule are catered by using milestones strategy and also by strict time management. When Component based SE was introduced the major pledge that it had over custom development was regarding time. By using COTS, time of coding and maintenance was highly reduced and left more time for requirements gathering, risk analysis and testing which in return has direct impact on Quality. A component selected based on requirements, will accelerate the SDLC to testing phase eliminating the time for designing and coding from scratch.

2.3. Complexity Comparison:

The discussion to use COTS or reuse a legacy Component cannot be made simply because the item "Fits In" the architecture [3]. Though architecture is a key to reuse [4], yet we must not overlook the result of above discussion that COTS products often require significant effort for their integration in a system. The complexity comparison supports Custom SDLC as inventing from the scratch is a simplest approach to be followed but there are chances that a person can take over a project in middle which can be troublesome. In all these scenarios our proposed system will be a great help because by that new comer can generate the report of the under construction module and can have a quick go through of what has been done and what is due to be done.

2.4. Issues with COTS:

Component based SDLC starts with requirements gathering which is extended to the selection of Components or COTS, then these components are learned and customized and then integrated to give the intended software which is then tested and deployed. It seems to be a quite straight forward process but this has some major issues which certainly want urgent attention because without addressing these issues progress in Component based will not be proceed further. Some of the issues are:

- Component must support the style of interactions of the system architecture in order to work together with other components.
- Choosing a wrong component may be more expensive than fixing problems in custom Software.
- There may be no source code available and no way to see the internal details of the component to customize it [5].
- Unavailable, Incomplete or unreliable documentation of the component [5] can add to complexities as well.
- An unpredictable Learning curve associated with the component [5].

Among all these issues our proposed system will address one concrete issue that is concerned with the understanding of the component in order to customize it. A tool that can take component as input and generate report based on that component will be a great assistance for developers.

PRACTICAL CONSIDERATIONS

This section describes the software System Design that will develop to achieve the above mentioned goals. This Design inputs a component and will generate a detailed report which will contain

- Index page for showing all modules in the component.
- Detailed report about every module in the component in separate chapters.
- Diagrammatical Data Flow between different modules of the component.

3.1 System Architecture:

Proposed system will take a component as input and will generate a report regarding its internal working. The report will be generated by using the parsed code of the component. We need a parser to parse and tokenize the source code. Then index page and the detailed report per module will be generated. In the end all these reports and the index page is consolidated into a single report of the whole component in Figure 2.

3.2. Index Page Generation:

For index page generation we will traverse the tokenized list which contains all the tokens of the parsed code and check for module names and keep them in a separate list. This list will be used to generate index page. We will count the page per module in detailed report section and also store them in a list containing module name and pages of its report. Then we will merge both list and generate index page which will show the starting page and the module name like a standard book index. Working is shown Figure 3.

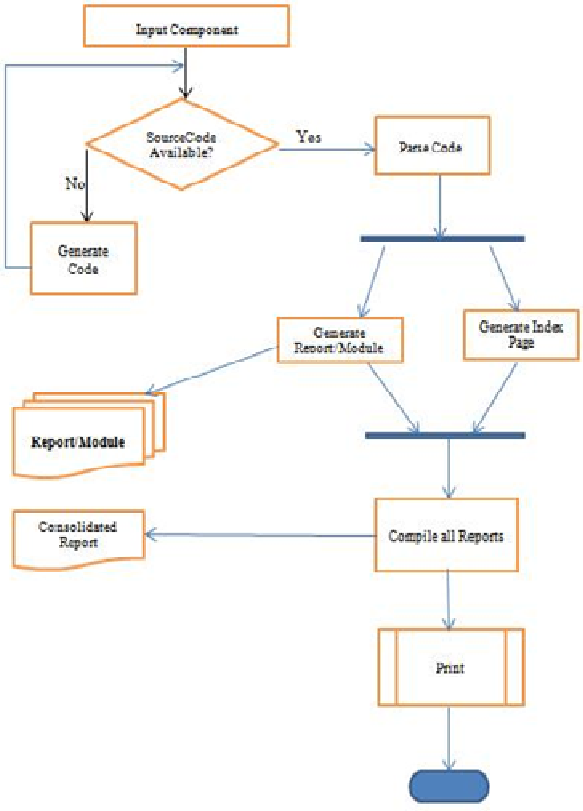

Figure 2: Complete System Flow

We are using three counters for this purpose. C_Page is a counter for current page. S_Page is a counter for the starting page of any module and E_Page is the end page counter of that module. The results of these counters are being stored in "n*3" size list where 'n' being the number of modules and '3' are the columns each for the respective counters.

3.3. Detailed Report Generation:

For detailed report per module we are using the approach of expected tokens. Tokenized and parsed code that is in the form of a list of tokens is used for this purpose. We will get a token; first decide whether it is a key word, variable, functions' name, class name or a number. Then we will get all tokens of one statement from the first got word till the next line is starting. We will locate the end of one statement by inspecting some end of line operators like"; "in C++.Then

we will use the tokens of each statement and concatenate them with some pre-defined phrases and form a sentence. This is how all statements of a module will be converted into English sentences.

> **For example:**
> **Input Line**: int abc =10;
> **Output Sentence**: *Variable* abc *is being initialized as* Integer *by a value* 10.

It is a very simplest example but can give a clear idea of Report Generation Schemas. The blend of phrases and keywords will give good structured sentences. The choice of phrase is done on the basis of tokens. As "abc" indicates that it is variable name as it is not a Keyword. It can be referred as a Function name but a function name has "("followed by the name. Likewise "=" indicated an assignment and "10"is its value. One more thing we will do, we will maintain a list of all modules and pages required in its report. This list will be used in Index page generation as shown in Figure 3.

3.4. Diagrammatical Dataflow Generation:

This data flow diagram is just a simple block diagram which will show the modules in the form of blocks. Input and output parameters of each module will be taken from the parsed list of code. These input and output parameters will be used to make a link list so that we can locate which module is taking input from which module. This diagrammatic flow will give a summarized idea of the whole report.

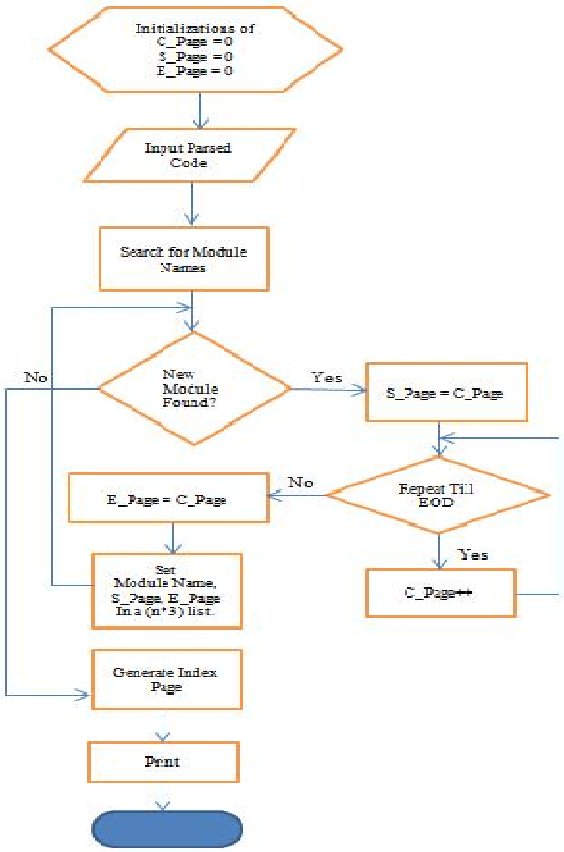

Figure 3: Index Page Generation

ANALYTICAL RESULTS

In order to check the usefulness of the provided paradigm a survey was conducted at 5 software houses at Islamabad among 30 Developers. The survey contained queries regarding the learning time of a component that is to be used in component driven development. Survey contained a checklist with learning time against KLOC (Kilo line of codes) of the component. Learning time is compared when no report is present across the component and when a report or documentation is present. The results showed that developers felt easy in understanding the component when the report of component was available. The results of the survey can be explained as in table 1.

Results show drastic decrease of learning time when the report of the component was available and the decrement can be considered up to 50%.

Table 1: Results for learning time with and without Report

KLOC (Kilo line of codes)	Learning time without report	Learning time with report
1 KLOC (Small Component)	1-2 weeks	5-8 days
2-3 KLOC (Average Component)	2-3 weeks	10-15 days
4-6 KLOC (Large Component)	5-7 weeks	15-20 days
More than 6 KLOC (Heavy Component like DDL)	9 weeks approx.	30 days approx.

CONCLUSION

In CBSE the learning time of the component is critical and can be effective enough to halt the development process and violate deadlines. This learning time can be reduced by using a report generation system that will give an idea about the inner details of the component in human equitable language that is easy to perceive and comprehend.

FUTURE WORK

This report generation system can be used for making generalized report of every programming language. System can be designed using Artificial Intelligence and Neural networks to make more generalized and intellectual reports. This system can be evolved to a much higher spectrum and can be used as a helping component in CBSE.

REFERENCES

[1] en.wikipedia.org/wiki/commercial_off_the_shelf

[2] Supriyo Bhattachariya, SDLC, College of Agricultural Banking, RBI Pune, India.

[3] Dan Galorath, Software Reuse And Commercial Off The Shelf Soft wares, President, Galorath Incorporated, EI Segundo, CA.

[4] Reifer, Donald J., (1997) Practical Software Reuse, New York.

[5] M. Morisio, COTS-Based Software Development: Processes and Open Issues, Politecnico di Torino, Torino, Italy.