Journal of Software Engineering and Applications, **2011, 1, 37-47**
doi:10.4236/jsea.2011.41005 Published Online January 2011 (http://www.scirp.org/journal/jsea)

Scientific Research

Testing Component-Based Software: What It has to do with Design and Component Selection

Shyam S. Pandeya, Anil K. Tripathi

Department of Computer Engineering, Institute of Technology, Banaras Hindu University, India
Email: pandeyashyam@gmail.com, anilkt@bhu.ac.in

Received November 26th, 2010; revised December 17th, 2010; accepted December 26th, 2010.

ABSTRACT

In a component-based software development life cycle, selection of preexisting components is an important task. Every component that has to be reused has an associated risk of failure of not meeting the functional and non-functional requirements. A component's failure would lead a developer to look for some other alternative of combinations of COTS, in-house and engineered components among possible candidate combinations. This means design itself can readily change. The very process of design of a software system and component selection seems to be heavily dependent on testing results. Instability of design, further, becomes more severe due to requirements change requests. Therefore, this instability of design has to be essentially mitigated by using proper design and testing approaches, otherwise, it may lead to exorbitantly high testing cost due to the repeated testing of various alternatives. How are these three activities: *Component-based software design, component selection and component-based software testing interrelated? What process model is most suited to address this concern? This work explores the above questions and their implication in terms of nature of a process model that can be convincing in case of component-based software development.*

Keywords*: *CBSE* (*Component-Based Software Engineering*), *Software Testing*, *Software Process*, *COTS* (*Commercial-Off-the-Shelf)

1. Introduction

Component based software development stresses reuse of preexisting in-house and COTS components [1]. This makes a developer look for the components that can be reused in a specific case. How to select a component that can be reused in a specific case? Traditional approach of selecting a component based on an architectural design of a software system is not effective [2]. A software system must be designed around existing components [2]. Any such design would incorporate preexisting components, and, those components, which are required to be engineered afresh. It may happen that after expending considerable effort in developing required components, some faults are detected in COTS or in-house components in terms of some mismatches, or, in form of some design flaws. Changing design or trying a different arrangement of components can only deal with this situation. This may considerably change the current design of the system under development. As a result, in order to test the new arrangement of components, all the previous activities of development of drivers, stubs etc. have to be accomplished afresh. This exorbitant cost of testing can be a big hurdle for success of CBS development, unless, it is dealt with efficiently. How to inculcate these concerns at process/process model level? What are the extra key engineering phases and sub-phases that must be incorporated in the present processes/process models. This work proposes a process for CBS development, which mitigates above-mentioned risk. Further, this work compares the proposed process with other CBS development processes/process models. Section 2 introduces CBSE. Section 3 discusses the nature of process model that can be convincing in case of CBS development. Section 4 proposes a process model for CBS development. Section 5 compares the model proposed in this work with other processes and process models in the literature. Section 6 discusses related work and section 7 draws conclusions.

2. CBSE

CBSE is a branch of software engineering which is concerned with development of software systems based on existing in-house and/or COTS components. Reusing previously developed components in developing software has many benefits. Important benefits are reduced cost and time to market [1]. Further, since a component is re-

Copyright © 2011 SciRes.

peatedly used, it undergoes repeated testing when used in different systems and operating environments. This will definitely increase the quality of a stand-alone component as well as the systems where it is being used. This can greatly help in materializing the benefits of standard domains. A standard domain can always provide well-defined standard components, which can be easily reused in any case. So, component technology heavily supports code reuse, which was previously not the case.

CBSE approaches software development in a way, which is very different from other previous approaches such as procedural and object-oriented approaches. In these approaches, often, the same team is responsible for developing the classes and procedures that develops a software system. This very approach is not very efficient in developing high quality large and complex systems in minimum time [1]. Further, procedural or object-oriented approaches for these systems often may lead to slippage of time schedule, and, failure of projects [1]. CBSE is supposed to be an effective approach for these big and complex systems. Peculiarity with CBSE is that it approaches software development in a way which is similar to that adopted by other popular engineering streams for manufacturing products. All of them develop products from highly reusable standardized smaller units (parts). For example, computer hardware is obtained by assembling different smaller parts like processor and other devices.

In the case of CBSE there are two separate development activities going on independently. Consequently, we essentially have two categories of developers: Component developers, and, system developers [1,2]. Component developers are engaged in development of components depending on market needs and domain expertise. System developers are engaged in development of software systems by assembling components. A System developer makes an effort to look into repositories for components during development of a software system. If components are not available in repositories, then, COTS components are searched. Lastly, if, second step also fails then team itself needs to develop required components.

3. Evaluating Engineering Processes/Process Models for CBS Development

An efficient process is central to success of any software development effort. There are several frameworks for comparing software processes. CMMI [3] is one such popular framework which groups the software processes into different categories. Here, we are not concerned with such an approach. Rather, we are interested in concerns, which are important to be taken care of (in terms of key phases and sub-phases) by a process/process model in order to efficiently develop a component-based software system. We want to judge various CBS development processes/process models with respect to these concerns.

3.1. Requisite Features of an Engineering Process for CBS Development

CBS Process models demonstrate that a CBS system must be designed around available components [2]. CISD model [4] stresses that CBS development should incorporate three major phases of component identification, evaluation and integration. The Process model does specify phases for component selection, software design and testing. But, it does not consider the coordination and interdependence a key phases and sub-phases that are required to select COTS components and design a CBS system. Further, it does not provide guidance regarding approach that can be taken for testing a CBS system. Any COTS component that has to be reused must be identified, evaluated and integrated to accomplish some system functionality. This was formally specified by CISD model [4] in 1997. This model stresses that CBS development should incorporate three major phases of component identification, evaluation and integration. The work stresses that in identification phase COTS components are identified on the basis of requirement specification. In evaluation phase, COTS components are evaluated for its fitness. In integration phase, all the development activities are accomplished, and, components are integrated. Many process models have been proposed apart for CISD model [4]. None of these process models provides insight regarding the nature of approach that can be efficiently taken up for software design, component selection and testing activities. The three activities considerably affect each other. A component can be selected, only, if, it can be fitted efficiently in the software architecture. This fitness has to be ascertained based on some verification and validation. A well-defined sequence of phases that can be used for above task can be a convincing process model. An important concern in this connection is failure of a COTS component at later stages of development life cycle. This does considerably change the existing design. This essentially means, often, design can change at any stage of testing leading to repeated design and testing effort due to failure of one or more selected COTS components in unknown way. When such a failure occurs, a developer tries another combination of COTS and engineered components. In order to mitigate this risk, component selection has to be intimately connected with design and testing strategies. Each of the above said three activities heavily affects the other two. They need to be coordinated taking previously mentioned risk into account. A criterion based component selection approach [5] proposes a criterion set that can be used to either se-

Copyright © 2011 SciRes.

lect a component or compare one component with another. This does help in comparing components regarding quality and performance of two are more components, but, it is not very useful for component selection as it does not consider the interdependence of component selection, design and testing, and, the way, candidate COTS components can be verified and validated for its fitness in a reuse context. Whenever we try to reuse a component, we should thoroughly test it for its fitness as early as possible to get convinced that it can reliably be used in present case as it heavily affects stability of design, and, the other components, that have to acquired or engineered. We can't select all the components before starting design as has been mentioned in CISD model [4]. In such a case, a component that is seemingly reusable in a case might not be a good choice, or, even might not be reused at all due to some mismatch or fault. Further, it is not possible to find all the reusable candidate components in absence of design unless the system is trivially simple. An approach is required which can be used to verify and validate COTS component fitness as early as possible. This should be based on verification and validation of architectural constraints and functionalities required by over all system. Design, component selection and testing should go hand in hand as these three activities are highly correlated. This makes it essential to go for design approach that breaks design into many less complex modules with well-defined interfaces. This not only helps in analyzing the non-functional properties of software early in development life cycle, but also, lets a developer to deal with a COTS component failure more effectively. An architectural centric approach which incorporates non-functional parameters like performance, modifiability, testability etc. can be an effectively taken up. This means an iterative development, wherein an iteration is comprised of design a little, select components, test completed functionality is the strategy feasibly applicable in component-based software development. Any component that has to be reused must be extensively verified for its suitability in accomplishing overall functional and non-functional requirements of a software system. This is extremely important activity that has to be performed. We can't verify all the functional and non-functional requirements by conducting testing because testing might not be immediately performed so extensively in practice. Secondly, some quality parameters like performance, modifiability and portability etc. may become the basis for a component selection, and, depend on architecture of the system, and, can most appropriately be performed by analyzing the architecture of a system. This is often the case with big, distributed and complex software systems. This means it is imperative to analyze the architecture (during component selection) in terms of constraints and nature of interaction between modules that must be there to meet quality and functional requirements. Therefore, design must be verified for these functional and non-functional requirements. This activity must precede the choice of alternative of combinations of components to be selected. In fact non-functional requirements should play a key role in designing architecture itself. This would not only make our system development governed by non-functional requirements but also helps us in rejecting some infeasible alternatives of combinations of components early in development life cycle. All these requirements have to be taken care of by the process required to engineer a component-based soft- ware in terms of nature of phases and sub-phases that have to be inculcated in the CBS development process/process model. We can arrive at the nature of process, which might be convincing by inculcating the support for the concerns discussed above. The following would have to be salient features of such a component-oriented approach for CBS development:

1) An iterative development approach incorporating activities like—design a little, select components and test completed functionalities, is a reasonable approach to be taken up. There is always a risk of failure associated with each component selected for integration. It might not function as required. So, each component selected must be tested at earliest so as to get convinced that it indeed supports the required functionalities. This will reduce rework in terms of repeated design, selection and testing.

2) A modular design approach is important to effectively use, verify and validate the COTS components.

3) It should concentrate first on important functionalities to be delivered by the system. It is obviously more applicable for large and complicated systems. Developers want to start testing important functionalities at earliest.

4) A process should drive architecture and design of a software based on existing components. COTS components do not come with their source code. They cannot be tailored much. It is application design, which is required to be developed based on existing components.

5) A component selection should not only be based on its own functionalities, but, also, the way it works in combination with other selected and engineered components.

6) Process should guide and let developer reuse existing components.

7) Component selected should be reused based on extensive verification and validation for its fitness in the current context.

8) It should help a developer negotiate for changes in requirement specification based on existing components.

9) Non-functional requirements like performance, modifiability, changeability etc. should be central concern of

Copyright © 2011 SciRes.

the design approach being taken up.

3.2. A Model for Comparing CBS Process Models

We propose a model based on the above discussion. The objective of such a metric model is to evaluate different process models that have been proposed for CBS development. We will use GQM paradigm to evaluate different process models. The Goal of evaluation is to check the fitness of existing process models against the features and attributes that must be incorporated in a process model that has to be used for CBS development. This goal must lead to the lower level goals. These lower level goals must lead to the questions that are required to be answered in order to answer the high level goal. These goals have to be obtained from the problems and difficulties that have been of concern for CBS development as has been discussed in the previous section.

From previous discussion, we have come up to the following high-level goal:

Efficient development of software systems using pre-existing software components.

It has to be expressed in terms of sub-goals, which must be accomplished to meet the above goal. We obviously take the lower level goals from the salient features that have to be supported by CBS development process. We come up with the following lower level goals.

- Iterative development support utilizing design a little, select components, and test completed functionalities as an iteration.
- Implementing important functionalities first.
- Process should guide design around available components in terms of phases and sub-phases required therein.
- Component selection approach based on analysis of support for functional and non-functional rquirements at following levels:
  - Component
  - Assemblage of components, which is affected by the component to be selected.
- Extensive verification and validation approach for evaluating the fitness of a COTS component in a specific reuse context by using architectural analysis techniques and testing approaches.
- Mechanism for requirement change negotiation.
- Architectural centric development approach which heavily stress modifiability, changeability, reusability and testability etc. as important quality parameters.

In order to measure above goals we need to answer whether a process/process model has following attributes:

- Iterative development
- Design a little, select a little and test a little feature
- Modular design strategy
- Important functions first
- Design approach which explicitly lets a developer identify reusable candidate components
- Coordination of design, component selection and testing (for mitigating risk due to a COTS component failure).
- Architectural centric analysis stressing, performance, changeability, testability and interoperability etc. as important design goals.

We want to evaluate our processes/process models around above questions. If a process does provide support for a certain feature we will define its measure as 1. If a process model does not provide support then we will use –1. In case, process model can be used for such support by further refining the process or process model, but, it is not visible in terms of phases and sub-phases present therein, we will use 0 as its measurement.

4. A Process Model for Designing and Testing Component-Based Software

This work proposes a use case driven integrated method for designing and testing a component based software. The process stresses the use-cases and software architecture as central assets for designing and testing CBS systems. Taking some use-cases at a time (in present approach) lets a developer concentrate on a smaller part of the whole system required to be developed, and, in the process, essentially, lets a developer divide the task into smaller less complex design targets. Further, this lets a developer to consider the functional requirements, non-functional requirements, and, available components, simultaneously, in a way, which is effective in reusing existing components. This approach by its very nature dictates the top down approach, which is obviously useful and reasonable in case of component-based software, as reusable components lets a developer access only component level details in terms of interfaces, quality, user documents etc. **Figure 1** diagrammatically represents the proposed process of CBS development.

Outline of such a process would be as follows:

1) Specify requirements of the software that has to be developed in form of use-cases. Also specify non-functional properties that have to be supported for each use-case. Certain properties might apply to all the use-cases.

2) Specify the high level modules based on system's interface with environment. Develop a high level module-oriented architecture of the system.

3) Categorize the use-cases in terms of their importance.

4) Use the architectural design technique to refine the high level architecture into lower level taking into account architectural styles, reference architectures involved sub-domains and non-functional properties of soft ware

Copyright © 2011 SciRes.

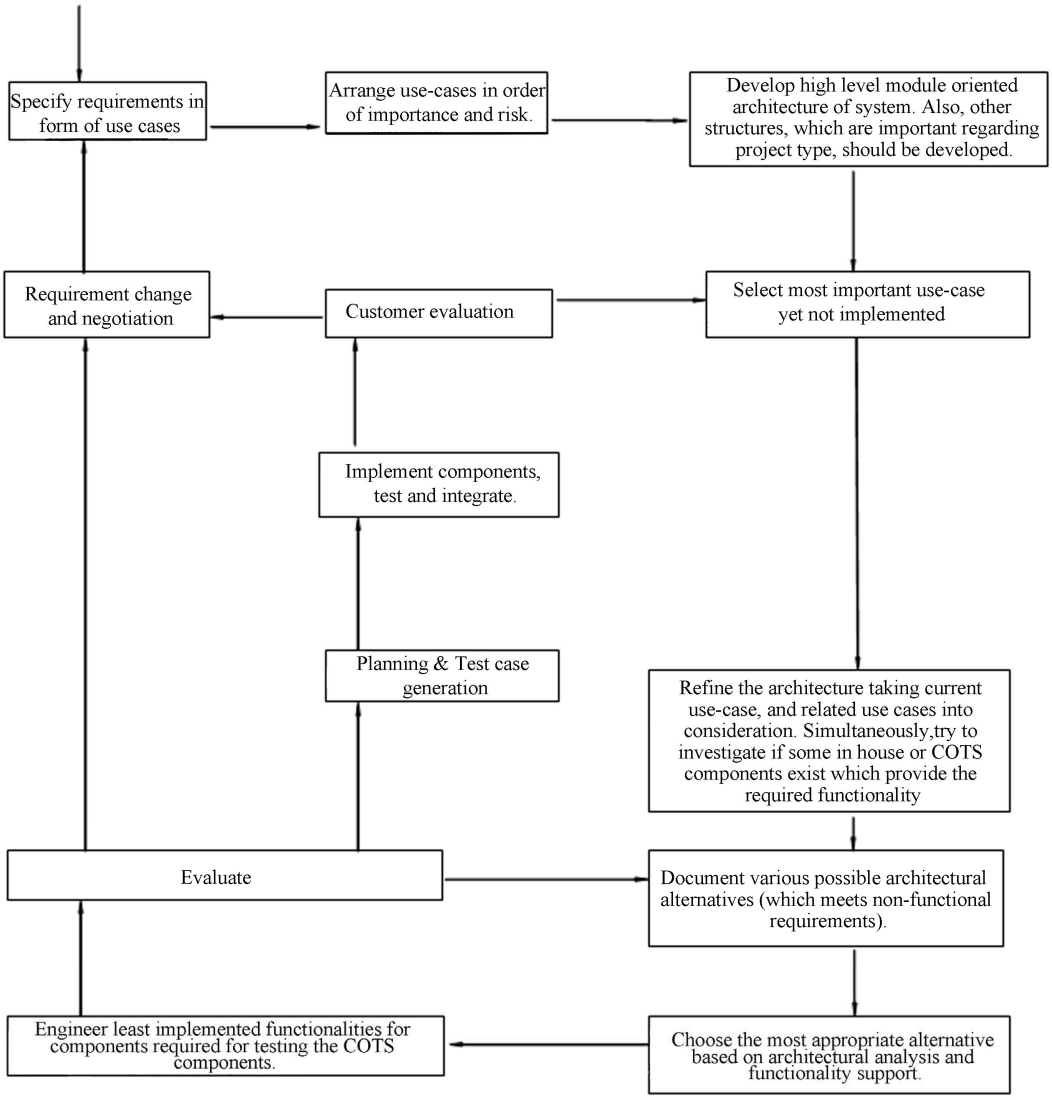

Figure 1. Software development process model for CBS systems.

into account. In each step map the functionalities required by use-cases to high-level modules. Continue this process until modules related to most important use-case (and related use-case) are separated out. Other modules not related to such a use-case might not be refined at this level necessarily.

5) Continue refining architecture based on most important use-case, and, those use-cases directly related to the former.

6) At every step of refinement try to search for components that can be reused in present case.

7) If some reusable candidate components are identified, various feasible architectures are documented.

8) These alternatives are rigorously analyzed for functional and non-functional supports based on documentation and partial architecture of the system.

9) Unacceptable alternatives are rejected.

10) Then those that are included should be more extensively analyzed by choosing criteria set that might not necessarily incorporate only technical factors. Component vendor reputation, experimentation facility etc. might be of interest.

11) This phase is meant for verification of chosen architecture and COTS components. This may incorporate experimentation facility from vendors, and, some implementation to test chosen architecture and/or COTS components.

12) If a failure is detected in the previous phase, the procedure is repeated for another available architectural alternative. If failure is due to a fault in a COTS component rather than design fault, then, the component vendor can be communicated so that component can be reused,

Copyright © 2011 SciRes.

if, the fault can be managed to be removed by component vendor.

13) If it is essential to change some requirement then the developer negotiates with customer for the changes in requirement specification.

14) Planning for component development, component integration order and testing is accomplished in this phase. Test cases are generated for integration and system testing based on some testing model.

15) Required components which are to be engineered are developed and integrated in this phase.

16) The above process is repeated for all the use-cases in order of their importance.

Start by specifying requirements of a software system in form of use cases. Analyze the requirements for their criticality and importance. Categorize the use-cases in different categories based on criticality and risk-proneness. Design a high level architecture of software, which must be supported by the software system. For example, if, requirements dictate access and modification of some large data set in a networked environment, then, there is always at least one database, and, clients, who can access and modify the data. This will let a developer have a perception of high level modules and corresponding architecture that might be required in the current specific case. If possible, divide use-cases in groups such that use-cases in one group are not connected with others in the sense that they need different components to accomplish the specified scenarios. Now, all these groups can be independently implemented with well-defined interfaces, if there is any. Take a use case highest up in hierarchy and try to document what functionalities are required for its implementation. Try to refine the existing architecture to get architecture alternatives, which can be used to accomplish this use case. This refinement may also take into account those use cases which seem to be highly related to this use case in terms of components required. Try to make a guess of components and their functionalities that might support the functionality dictated by that use case and resultant architecture. Use words for functionalities or from concerned domain to search components. Components guessed should also be searched. Scan various results for the main functionalities they support and analyze their functionalities, and, find, if they can be integrated with some home developed components to support the functionalities. Document various feasible component combinations that can provide the functionalities. Incorporate these components combination alternatives into the original architecture to change and refine this existing architecture. Collect other use cases from the list, which directly needs these components for accomplishment of its functionalities. Make an analysis of these use-cases that directly need the required functionalities from components that are there in refined part of the given architecture. Analyze the various alternatives for non-functional requirements and performance. Those alternatives are found which are useful for their support for implementing functional and non-functional requirements for the use case that has been taken up, and, the related use cases. One of these alternatives is required to be chosen. Starting with most convincing architecture, we must experiment various COTS components specified there for understanding the functionalities supported by them. This lets a developer understand COTS components effectively. Test cases should be developed for testing COTS component and/or non-functional requirements (with minimum development effort), if required. We can't afford to design all the required components before getting convinced that COTS components indeed can be effectively be used in current context. The idea is to make a part of the architecture stable, which is supposed to be independent from others. This architecture should be based on available components and their support for functional and non-functional requirements. Once the architecture becomes stable based on above experimentation, concerned COTS components are required to be integrated with other components, which are required to be engineered. The developer may need to negotiate in case of some mismatches in functionalities supported by components, and, that required by use-cases. In practice, stakeholders are more concerned with the cost of software even if that may lead to some changes in their original requirements. Next, planning for development and testing activities is done. Test cases are generated based on some model of the system (part being implemented). These test cases can be used for integration testing as COTS and freshly engineered components are integrated. This whole process is repeated for all the use-cases starting from most critical use cases.

5. Comparing the Proposed Process Model with Processes and Process Models Reported in Literature

It is interesting to analyze the previously proposed processes/process models regarding their support for CBS development. None of the process models stress the coordination of design, selection and testing. This means, these process models do not provide much help regarding this vital risk of volatile nature of the design in case of CBS development. Our process model explicitly specifies the approach that can reasonably be applied to mitigate this important risk in CBS development. Our process model specifies key phases that can efficiently be used for 1) Designing a CBS system by using preexisting in-house and COTS components, and, 2) Verifying and validating a CBS system. Below, we compare proposed

Copyright © 2011 SciRes.

model with some important CBS centric process models.

5.1. CISD Process and the Proposed Process

CISD process model [4] incorporates three broader phases:

The product identification: Major activity during this phase is 1) Requirements analysis and classification 2) Product identification and classification 3) Product prioritization. COTS product identification also updates overall architecture of a system. This approach mandates identification of all the components and corresponding architecture in this phase. This approach practically is not feasible. Problem is that COTS components for big and complex system cannot be identified based on requirements only. Design is essential to know various reusable COTS components. So, design and component selection has to go hand in hand. We need to start designing architecture based on well-defined quality parameters, and, simultaneously, need to look for possible reusable candidate components. It is not practically possible to search COTS components first and then design the architecture based on that. In fact, analysis of architecture is essential to know about the components that can be used. This problem does mandate design a little, select a little and test a little approach as has been incorporated in current proposed process. Architectural development neither can be delayed till component selection nor it can be accomplished before component selection. Architectural design and component identification should be accomplished simultaneously.

The product evaluation: In this phase product (component) is evaluated for functionality, architecture and performance. This approach seems plausible but practically can lead to very high cost of evaluation and rework in terms of redesign and evaluation. Concern for some non-functional requirements should start early during architectural development. Architectural development must be governed by well-defined quality and performance parameters. Further, development approach should stick to design a little, select components, and test a little approach.

The product integration: In this phase all the development work required to accomplish integration of COTS components is accomplished. This work of big-bang integration is the central problem that has been addressed in this work. This would certainly make CBS development costly affair.

CISD model adequately brings forth the nature of key activities of CBS development. But, design, component selection and testing activities have not been coordinated in a way, which can explicitly be followed to develop a CBS system. The approach does not mitigate the problem of instability of CBS design as discussed previously. It does not stress the risk of a COTS component failure at later stages of development life cycle. Nature of verification and validation approach that can feasibly be taken up is not explicit. This, lack of proper verification and validation process, is one of central issues in CBSE. Process models do not provide sufficient insight regarding the verification and validation approach that can be taken up.

5.2. COSE and the Proposed Process

Component-oriented software engineering (COSE) process model [6] consider a structural decomposition for arriving at existing components. It starts with system specification. Then Decomposition is accomplished. In the next phase, components are specified, searched, modified and created. Then, components are integrated to get overall software system. Problem with this approach is that this process model too doesn't sufficiently stress the concern of instability of design due to the failures of COTS components at later stages of development. It does not address the instability that can result using such an approach. It doesn't make explicit the short of verification and validation approach that can be taken up.

5.3. A Reusable Software Component-Based Development Process Model and the Proposed Model

This is a spiral model of software development proposed in 2007 by M. R. J. Qureshi and S. A. Hussain for CBS development [7]. This model is very similar to reuse spiral [2] proposed in 1997. Spiral in this process model consists of following phases

- Communication
- Planning
- Analysis and component selection
- Engineering and testing
- Evaluation

In this process model, analysis and component selection phase is meant for analyzing requirements for selecting COTS components that can be used. This approach can work for developing applications similar to that developed in past. In a new system it is not possible to identify components based on requirements only. Architectural design and component selection must go hand in hand. This process model does not incorporate phases and sub-phases required for experimenting and evaluating components. Model does not sufficiently closely consider CBS centric concern and risk of COTS component failures. It does not provide the insight regarding:

1) Design approach that that can be taken up for maximizing reuse of existing components.

2) Component selection approach.

3) Verification and validation approach that can be taken up.

Copyright © 2011 SciRes.

There are other process models too but they do not mandate anything except components should be selected before completion of design phase. A process for CBS development must incorporate the evaluation of components for their fitness in terms of well-defined activities. In fact, certain problems like component trust problem [1] is direct result of our inability of devising verification and validation strategy that can be taken up for ascertaining the fitness of COTS components in a specific case. This problem becomes more sever because of the failure of a COTS component which can lead to the rework in terms of repeated design and testing effort. In **Table 1**, we compare the process proposed with other CBS processes/process models.

CBS process models do not explicitly specify the key phases and sub-phases required to develop a CBS system. CISD model does specify activities that have to be incorporated in a CBS development, but fails to provide sufficient support for CBS design, component selection and concern of COTS component failures in later stages of development cycle. This is an important concern that has been taken care of by proposed process model. What is required is explicit phases and sub-phases that can be followed to develop a CBS system. One phase should logically and smoothly lead to the next one. Previous process models do not provide such support. They do not sufficiently guide the way a CBS system can be verified and validated. Proposed approach does provide insight regarding these problems. Except third and fourth process models, other process models do not specify the reuse-oriented design approach that can be used for maximizing COTS components reuse. They do not specify phases and considerations that have to be essentially incorporated in order to design a CBS system in presence of available components.

Our process inculcates the important concerns that must be incorporated to select components and design software by specifying key activities and phases, and, their appropriate ordering. Further, this approach if taken up can greatly reduce testing effort, which, otherwise, might be a big hurdle to successful CBS development.

6. Related Work

An important task in a CBS engineering process is component selection. Unless, we can incorporate this concern effectively in our engineering process, it is hard to engineer CBS systems using preexisting components. A wide variety of components for one and the same purpose may be available. One has to select right components for a purpose under constraints that they have to be interfaced with other components. Further, the assemblage of reused and engineered components should support the software requirements. In traditional reuse approach, developers are encouraged to look for needed components after most of the design work has been done [2]. Rather than waiting until design is done (and then looking in vain for matching components), software products need to be designed around available software components [2]. This is called Reuse-driven development [2]. Reuse driven development [2] dictates reused component selection early in the development life cycle. Further, these components not only affect design phase but also can modify system specification [2]. Process model doesn't provide phases that might be required for component selection. Johannes Sametinger proposes reuse spiral [2], which specifies and tailors Boehm's spiral as per requirements of CBS development.

Table 1. Comparison of the proposed process model with other CBS process models.

Feature / Process/ Process Model	Iterative development	Design a little test a little approach	Important function first	Modular Design	Design approach which explicitly a let developer find the reusable candidate components	Coordination of design, component selection and testing	Architecture centric analysis (and its early validation by testing)
Reuse driven process [2]	–1	–1	–1	1	–1	–1	–1
Reuse spiral [2]	1	0	1	1	0	0	0
CISD model [4]	1	–1	0	1	–1	–1	–1
Process model by Ali H. Dogru [6]	–1	–1	–1	1	1	–1	–1
Process model by J. Gao [1]	–1	–1	–1	1	1	–1	–1
Spiral process model [7]	1	0	1	1	–1	–1	–1
Proposed process	1	1	1	1	1	1	1

Copyright © 2011 SciRes.

Author asserts that reuse spiral allows explicit and early consideration of reuse by identifying alternate means for the implementation of components of the system to be built. This model doesn't consider component selection phases, and, verification and validation approach that can feasibly be applied. Vu Tran and Dar-Biau Liu in their work [4] conclude that Waterfall model and Spiral model fail to reflect procurement-centric nature component-based software engineering process. The paper asserts that key engineering efforts in software integration process include:

- Classifying and deriving domain requirements for early COTS product selection.
- Partitioning system into sub-domains for COTS product evaluation.
- Defining architecture based on selected COTS product architecture
- Defining strategy and criteria for COTS product evaluation
- Identifying and prioritizing candidate COTS product.
- Evaluating prioritized COTS products.
- Integrating COTS products combination in final system.

Further, the work proposes a COTS-based Integrated System Development (CISD) model by inculcating these key activities. Model stresses on selection and integration activities. The CISD model consists of three distinct phases: Product Identification, Product Evaluation, and Product Integration. Process model doesn't stress on the interdependence of design, selection and testing. It is not possible to identify and select all the components beforehand. It has to be accomplished hand in hand with design and testing activity. A COTS component needs to be evaluated in presence of design and functionality requirements. Independent evaluation is not of much use. Proposed process explicitly makes apparent the phases of evaluation that can be done in a specific case. The CISD model is not providing this very crucial support. Vu Tran, Brad Hummel and Dar-Biau Liu in a work entitled "Understanding and Managing The Relationship Between Requirement Changes and Product Constraints in Component-based Software Projects" [8] asserts that change requests drastically change previous combination of components used for implementing the functionality. This forces repeated rework, as new components combinations are required to be evaluated every time. This has to be mitigated effectively as it can severely hamper the progress of development effort leading to low quality product. Hans-Gerhard Gross and Nikolas Mayer in a work entitled "Built-In Contract Testing in Component Integration Testing" [9] puts forth the idea of built-in test to reduce the manual effort required to evaluate the conformance of one component from others which are supposed to be interfaced from it. Ali H. Dogru and Murat M. Tanik in a work [6] assert the investigation of reuse eventually matured into the "build by integration" paradigm. Component technologies improved along with engineering practices, but, they lack a methodology that uses components within such a paradigm. Further, the paper proposes a generic process model for CBS development. Process model does not consider the volatility of design as an important risk. This makes this model inappropriate for CBS development. Jerry Zeyu Gao, H.-S. Jacob Tsao and Ye Wu in a book entitled "testing and Quality Assurance for Component-Based Software" [1], proposed a process model which stresses that design has to be done around available components. This model also doesn't provide any support for component selection, software system design and testing activities. A work entitled "A COTS Architectural Component Specification Stencil for Selection and Reasoning" [10] asserts that selection and assessment of COTS components are still a challenge task. It is hard to find the right components that exactly fit into the requirements. The selection processes are in general ad-hoc. Any wrong choice of COTS components may compromise the benefits from reusing these components since the chosen component may mismatch with other components and the environment. A paper entitled "Measuring the usability of software components" [11] asserts that one of the most critical processes in CBSD is the selection of a set of software components from in-house or external repositories that fulfill some architectural and user-defined requirements. However, there is a lack of quality models and metrics that can help evaluate the quality characteristics of software components during this selection process. A paper entitled "Software Component Specification: A Study in Perspective of Component Selection and Reuse"[12] asserts The current best practices for component specification ignore information that is vital in determining if an available, ready to use component, contains precisely the functional and extra functional properties required, and, if, that component can be used in the target environment. One paper entitled "Determining criteria for Selecting Software Components: Lessons learned" [13] asserts that component selection relies on suitability and completeness of criteria used for evaluation. Another paper entitled "Approximation Algorithms for Software Component Selection Problem" [14] also make similar assertion that selecting a set of components to satisfy a set of requirements while minimizing cost is becoming more difficult. A work entitled "A reusable software component-based development process model" [7] proposes a spiral process model for CBS development.

Above discussion shows that most of the papers stress

Copyright © 2011 SciRes.

that component selection is costly affair. An efficient component selection is pivotal to CBS development. But, they don't sufficiently stress that component selection should not be viewed in isolation. It must be coordinated with design and testing activities. They do not provide guidelines regarding the phases and nature of activities required therein. Our work stresses the coordination of these activities and provides a feasible approach that mitigates the important risk associated with instability of design. It explicitly brings forth the nature of verification and validation approach that can be taken up. This was not apparent in previous models. Further, the design approach stressed in proposed model lets a user easily look for the kind of a COTS component that can be reused, and, the way it has to be verified and validated for its fitness in a specific case. The new process model brings new issues. For example, new verification and validation strategies are required which can be used to validate the reuse of reusable components during architectural design. These validation models should stress on contextual verification of reusable COTS and in-house components. New model will have weaknesses if these issues are not addressed properly.

7. Conclusions and Future Work

In case of CBS development there can be many choices possible from amongst the components for a particular situation. Not all of these possible choices can appropriately fit into the system guaranteeing perfect integration with full functionality and performance. Such a situation calls for testing of multiple components at the time for ensuring their fitness for functionality and performance requirements. The previous Process models in CBS development do not explicitly specify the key phases and sub-phases which essentially clarify the way component selection, CBS design, and, testing are dependent on each other, and, the nature of key activities that are required to be accomplished during those phases. This understanding is essential to devise an appropriate CBS development approach. An important concern is the failures of COTS components during later stages of development. This can lead to increased integration and testing cost making CBS development unprofitable undertaking. This work identifies the role and place of testing, and, other related activities in the CBS development process model. It further proposes a software development process model, under CBSE, that appropriately integrates the key design, selection and testing activities at suitable places. Finally, this work compares proposed model with other models available to indicate the characteristics that the proposed model does have as opposed to the other models. Important characteristics that the proposed model does possess, whereas other models may be lacking, are the following:

- It mitigates the risk of COTS component failures at later stages of development life cycle by specifying key phases and activities therein for verifying and validating the fitness of a COTS component in a reuse context.
- It brings forth the role and place of architectural analysis, component evaluation and testing in CBS systems.
- It brings forth the nature of interdependence in design, component selection and testing activities in case of CBS development.

These understanding can help in devising proper methods for accomplishing CBS centric analysis, design and testing activities, and, can be pivotal in reducing verification and validation effort required therein.

In order to apply the proposed model, we need to develop new techniques for various activities during the various phases of software development. We need models to verify architecture early in development phase. Further, models for verifying COTS and reusable in-house components reuse at architectural design phase are required.

REFERENCES

[1] J. H. Gao, H. S. J. Tsao and Y. Wu, “Testing and Quality Assurance for Component-Based Software,” Artech House, Norwood, 2003.

[2] J. Sametinger, “Software Engineering with Components,” Springer-Verlag, New York, 1997.

[3] C. Ebert and R. Dumke, “Software Measurement,” Springer-Verlag, Berlin, 2007.

[4] V. Tran and D. Liuo, “A Procurement-Centric Model for Engineering Component-Based Software Systems,” 5*th International Symposium on Assessment of Software Tools* (*SAST*'97), Pittsburgh, 2007.

[5] J. P. Carvallo, X. Franch and C. Quer, “Determining Criteria for Selecting Software Components: Lessons Learned,” *IEEE Software*, Vol. 24, No. 3, 2007, pp. 84-94. doi:10.1109/MS.2007.70

[6] A. H. Dogru and M. M. Tanik, “A Process Model for Component-Oriented Software Engineering,” *IEEE Software*, Vol. 20, No. 2, 2003, pp. 34-41. doi:10.1109/MS.2003.1184164

[7] M. R. Qureshi and S. A. Hussain, “A Reusable Software Component-Based Development Process Model,” Advances in Software Engineering, Vol. 39, No. 2, 2008, pp. 88-94. doi:10.1016/j.advengsoft.2007.01.021

[8] V. Tran, B. Hummel, T. A. Le, J. Doan and D. Liu, “Understanding and Managing: The Relationship Between Requirement Changes and Product Constraints in Component-Based Software Engineering Projects,” *Proceedings of the Thirty-First Annual Hawaii International Conference on System Sciences*, Hawaii, 1998.

Copyright © 2011 SciRes.

[9] H. Gross and N. Mayer, "Built-in contract testing in component integration testing," *Electronic Notes in Theoretical Computer Science*, Vol. 82, No. 6, 2003, pp. 22-32. doi:10.1016/S1571-0661(04)81022-3

[10] J. Dong, S. Yang, L. Chung, P. Alencar and D. Cowan, "A COTS Architectural Component Specification Stencil for Selection and Reasoning," *ACM SIGSOFT Software Engineering Notes*, Vol. 30, No. 4, 2005, pp. 1-4. doi:10.1145/1082983.1082959

[11] M. F. Bertoa, J. M. Troya and A. Vallecillo, "Measuring the Usability of Software Components," *Journal of System and Software*, Vol. 79, No. 3, 2006, pp. 427-439. doi:10.1016/j.jss.2005.06.026

[12] C. J. M. Geisterfer and S. Ghosh, "Software Component Specification: A Study in Perspective of Component Selection and Reuse," *Fifth International Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems*, Washington, 2006.

[13] J. P. Carvallo, X. Franch and C. Quer, "Determining Criteria for Selecting Software Components: Lessons Learned," *IEEE Software*, Vol. 24, No. 3, 2007, pp. 84-94. doi:10.1109/MS.2007.70

[14] N. Haghpanah, S. Moaven, J. Habibi, M. Kargar and S. H. Yeganeh, "Approximation Algorithms for Software Component Selection Problem," *Software Engineering Conference* (*APSEC* 2007), Japan, 2007. doi:10.1109/ASPEC.2007.38

Copyright © 2011 SciRes.