

Component-based Integrated Systems Development: A Model for the Emerging Procurement-centric Approach to Software Development

Vu Tran
ArcQuest Corporation

Abstract

The continuing increase of interest in Component-based Software Engineering (CBSE) signifies the emergence of a new development trend within the software industry. Unlike preceding software engineering models, CBSE heavily relies on the utilization of commercial off-the-shelf (COTS) products as the underlying foundation for new product development. Its emphasis is on the acquisition of reusable products to develop complex integrated solutions over their development from scratch. Compared to traditional development-centric approaches, CBSE promises a more efficient and effective means to deliver software solutions to the market. However, underestimating the risks associated with the acquisition of these software components has resulted in longer schedule delay and higher development/ maintenance cost. This paper describes a procurement-centric model that we have utilized to effectively support the development of a CBSE project at the Mitsubishi Consumer Electronics Engineering Center (CEEC). The Component-based Integrated Systems Development (CISD) model identifies key engineering phases and their sub-phases that are often ignored, or merely implicit, in existing development-centric models. The paper also presents the various lessons learned implementing CBSE at the CEEC.

1. Introduction

The prospect of increased product reliability and stability, at shorter development time and reduced cost, has led to a recent surge of interest in CBSE. This engineering approach emphasizes the acquisition and integration of reusable COTS products over in-house development for constructing complex and large-scale software solutions. At the CEEC, CBSE has received

very close attention early on. It has been chosen as the approach for developing the DiamondWeb^{TM2} Internet/Television product family. The DiamondWeb project is an ambitious attempt by Mitsubishi to provide fully integrated Internet/Television solutions for the consumer electronics market in North America. Some technologies chosen for the end-product include Java, World-Wide-Web, TV HTML, Netscape-like plug-ins, Netscape Secured Socket Layer (SSL), TV image and Web HTML printing, Web page zooming and panning, and many others. Future technologies that are being investigated include digital television, and plasma display television. The first shipment for North America consists of all the Internet technologies mentioned above, embedded in existing large-screen Mitsubishi television product families.

The key reasons behind the selection of CBSE as the approach to engineering the DiamondWeb system include:

- 1) To meet a demanding development schedule.
- 2) To reduce the need for engineering specialists to support the development of the various technologies.
- 3) To leverage the reliability of existing COTS products.

Recognizing the limitations of existing development-based engineering models [TRAN97b], a procurement-centric model was developed and adopted as a framework to support both project planning and product development. The CISD model describes a systematic approach to development of integrated software solutions using commercial off-the-shelf components [TRAN97a]. Unlike development-centric models, the CISD model is designed specifically to reflect the development steps associated with the development of COTS-based integrated systems.

In the next section, the paper summarizes the CISD model, and its integration with a development-centric model, used in the DiamondWeb project. Finally, the paper presents the lessons learned regarding CBSE from this project.

2. Analysis of the COTS product evaluation, selection, and integration process

The key engineering efforts often encountered in a software integration process include:

1. Classifying and deriving domain requirements in order to drive the early COTS product selection effort.
2. Partitioning the system into domain-specific subsystems to enable concurrent COTS product evaluation.
3. Defining an overall system architecture that is based on the architectures of the selected COTS products.
4. Defining an overall COTS product evaluation strategy and criteria that reflect the available time and resources.
5. Identifying and prioritizing candidate COTS product sets for evaluation.
6. Evaluating the prioritized candidate COTS product sets.
7. Integrating the selected candidate COTS product combination into the final system.

2.1 Classifying and deriving domain requirements

The evaluation and selection of COTS products can not be effectively implemented without understanding the requirements associated with the system to be developed. However, system requirements often do not directly address the characteristics of a particular application or service domain that are needed in COTS product evaluation. Rather, these requirements often describe the functional, performance, and interoperability needs of the entire system from a “black box” perspective. In CBSE, understanding of the overall system requirements is often not enough. These system-level requirements need to be broken down, extracted, derived, and organized into collections of domain-specific requirements. This effort is necessary to separate the system’s overall requirements into domain-specific sets to support subsequent concurrent product evaluation and selection efforts. This requirement derivation and classification process is not addressed in existing software engineering models, yet often required in component-based software development projects.

2.2 Partitioning of the system into domain-specific subsystems

Modern methodologies emphasize the importance of partitioning a large-scale software system into multiple manageable subsystems for concurrent analysis, design and implementation. To ensure maximum internal

cohesion and minimum external coupling between these subsystems, the partition should reflect the various application and service domains associated with the system [AWAD96]. This effort is imperative to support early identification of candidate COTS products for evaluation. More importantly, it also enables early identification of subsystems that cannot be supported by the procurement of COTS products. These subsystems will be implemented using a more development-centric approach. Figure 4 illustrates an example of the domain-based partitioning of those portions of the DiamondWeb™ system¹ that resides in the Mitsubishi television. The shaded boxes identify subsystems that can be implemented totally or partially by the procurement of COTS products. The white boxes identify subsystems that will be developed entirely in-house. The stand-alone boxes are generic utilities or services that will be used by other subsystems as needed.

2.3 Defining an overall system architecture

Defining consistent, extensible, and robust software architectural solutions continues to be challenging in software engineering. In CBSE, this task is much more difficult. First, the architecture of a component-based system is often influenced significantly by those of the selected components [CLEM95] [TRAN97a]. Thus, the selection of different software components, resulting from product evaluation and selection, has a significant impact on the overall architecture of the system. As a result, the final architecture of the system to be developed often continues to evolve throughout the COTS product evaluation effort. Second, as described in [GARL95], overcoming architecture mismatch among selected COTS software products is another challenging task often not experienced in development-centric software engineering. This is due to

- 1) assumptions about the nature of the product.
- 2) assumptions about the nature of the product interfaces.
- 3) assumptions about the overall system architecture.
- 4) assumptions about the product’s construction process.

As a result, the architecture of the selected products, and the protocol, control, and data mismatches between them, often provide additional challenges to the overall system architecture design effort in CBSE.

¹ Many details have been omitted to protect company proprietary and confidential information.

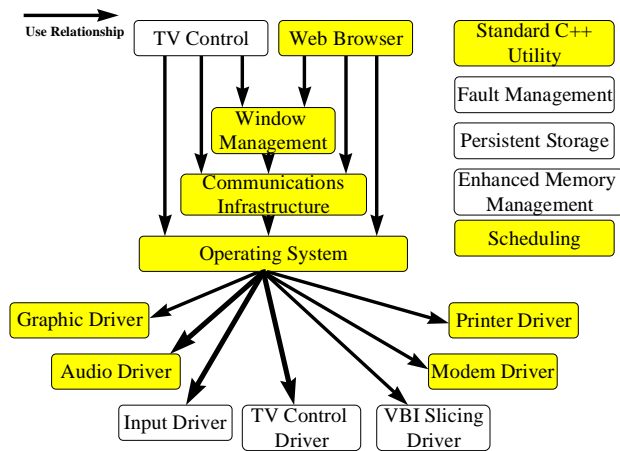


Figure 3 The DiamondWeb Internet/TV subsystems

2.4 Defining an overall COTS product evaluation strategy and criteria

With the increasing availability of COTS products, evaluating them is an endeavor demanding considerable time and energy. Furthermore, there are often no obvious winners among the candidate products that are being evaluated. Selecting an appropriate product typically requires tradeoff analysis among the available products in which only a “better than others” solution is available. As a result, establishing the criteria for the selection of these products is a very important task in CSBE. These product evaluation and selection criteria always extend beyond those defined by the system functional requirements. Some criteria are concerned with the product vendors, such as: 1) the level of product support expected for the life of the system, and 2) the stability of the company. Others reflect the limitation of time and resources to support the evaluation effort, such as: 1) the deadline associated with the final product selection decision, 2) the resources that can be allocated for the evaluation effort, and 3) the number of products to be evaluated. Large-scale integrated system development often requires concurrent evaluations of multiple software domains, with multiple engineers across multiple development organizations. As a result, early establishment of these criteria provides a uniform guideline for timely and consistent evaluation of COTS software products.

² OpenView® is a registered trademark of Hewlett-Packard Corporation

³ RTWorks® is a registered trademark of Talarian Corporation.

⁴ Orbix® is a registered trademark of IONA Technologies Ltd.

2.5 Identifying and prioritizing candidate COTS product sets

The COTS product identification phase typically requires extensive effort and time spent in attending conferences, reviewing literature, training, traveling and communicating with product vendors. Unfortunately, current development-centric models do not explicitly recognize these as normal parts of the development effort. In procurement-centric development, the efforts to support COTS product identification often require a large amount of dedicated human resources and time. In addition, the selection of an inappropriate candidate product for integration can result in an enormous amount of extra time and effort to re-evaluate and re-implement the system with another product. To support realistic cost analysis and project planning, the process of identifying candidate COTS products should be captured as part of the component-based software development.

During a product identification effort, an important criterion for early elimination of candidate products is their inherent dependency. Products that depend on others which violate the system requirements can be eliminated from the evaluation effort early. For instance, products that do not work with the required operating system should be rejected.

While identification of appropriate candidate products for evaluation is an important task, *prioritizing* them enables subsequent “on-schedule”, and often time-optimized, product evaluation. As described in subsection 3.4, the criteria used for prioritizing the candidate products are often extensive and subjective. They are dependent on many non-technical issues such as product costs, available time, product preference, product vendor’s history, etc. From a technical perspective, two important guidelines for prioritizing candidate products for evaluation include (1) the product’s ability to inter-operate with other products, and (2) its ability to support multiple identified application and service domains. Selecting products that are readily inter-operable enable reduction in the overall integration time and effort. In contrast, selecting a group of products that are not readily inter-operable means additional time and effort will be needed for product integration. Products that can support multiple domains should receive higher priority since they enable a reduction in the number of inter-product interfaces that need to be interconnected. Products that provide maximum support for both criteria above are often those that provide the widest coverage of the system requirements at minimum additional development cost.

In a system integration project, there are often multiple software products selected for a concurrent and

integrated product evaluation effort. This approach is needed in order to support selection of an optimal COTS product combination for the system to be implemented. From our experience, in the short-term, the limitations of a particular product might not be as important as its ability to collaborate with others to perform the required system-level functionality. In the long-term, however, the limitations of individual products can negatively impact the extensibility of the entire integrated system. As a result, the evaluation effort needs to address all candidate software products' capabilities as individuals and in collaboration.

2.6 Evaluating the prioritized candidate COTS product sets

The typical process for evaluating a COTS product includes:

- 1) acquisition of the product and its dependent products,
- 2) design of the appropriate prototype and test plan,
- 3) configuration of the development environment to support the product evaluation,
- 4) installation of the product and other dependent products,
- 5) actual coding of prototype,
- 6) evaluation of the product, and
- 7) generation of the appropriate product evaluation report.

In [BROW96], Brown and Wallnau provided additional discussions of these activities to support a software product evaluation. For CBSE, the level of complexity regarding COTS product evaluation increases significantly since multiple products will have to be evaluated in combination.

In general, there are three important areas in COTS product evaluation. These include functionality, interoperability and architecture, and performance. Functionality evaluation encompasses both evaluation of functionality of the individual products and of their integration. Interoperability and architecture evaluation ensures that all candidate COTS products can be integrated according to their product specifications. Performance evaluation addresses the performance of the integrated COTS products in supporting system-level functional threads. During this evaluation process, adequate prototype software is implemented to support all necessary integration and testing. Although the role of software prototyping has always been recognized as an important part of modern software development processes, software prototyping in CBSE takes on a much more important role: it is the only mechanism available for discovering the capabilities and the limitations of

candidate COTS products within the context of the system to be developed.

2.7 Integrating the final selected candidate COTS product combination

Once a set of candidate COTS software products has been selected, the development effort enters the product integration phase. This phase consists of the following efforts:

- 1) Developing the software adapters needed to interconnect all the selected COTS products,
- 2) Developing the necessary enhancements to these products,
- 3) Developing other parts of the system that can not be supported by available COTS products, and
- 4) Integrating and testing the final system.

Since these efforts often require in-house development and testing, a development-centric approach such as those that are derived from the Waterfall or Spiral model should be utilized. A collection of development-centric software engineering approaches is described in [McCO96].

3. Introducing the Component-based Integrated Systems Development (CISD) Model

The CISD model consists of three distinct phases: Product Identification, Product Evaluation, and Product Integration. The *Product Identification* phase includes the process of collecting and understanding the overall system requirements, identifying and classifying COTS product into product sets, and prioritizing them for the subsequent evaluation. The *Product Evaluation* phase includes the process of integrating, evaluating, and comparing these product sets to select the most optimal combination for integration. The *Product Integration* phase includes the building of all necessary software adapters and enhancements to the selected COTS product set to implement the required integrated system. The overall product selection, evaluation, and integration process is often iterative in nature, depending on the size of the system to be developed and the engineering resources available. Subsequent sections describe each phase in more detail.

3.1 The Product Identification phase

The Product Identification phase includes all technical activities that are required to generate a prioritized collection of product combinations for subsequent

evaluation. The major activities during this phase include (1) Requirement Analysis and Classification, (2) Product Identification and Classification, and (3) Product Prioritization. The **Requirement Analysis and Classification** stage encompasses the process of understanding the system requirements and partitioning these requirements into various application and service domains. The **Product Identification** stage encompasses the process of collecting information on candidate COTS products and grouping these products into different product combinations, or sets, for further evaluation. Subsection 3.5 describes various activities implemented during this stage, including the identification of subsystems that have to be built from scratch. These subsystems will subsequently be implemented using a development-centric software engineering method. The **Product Prioritization** stage includes the review of all candidate product sets to generate a prioritized list for further evaluation. The outputs of the initial COTS product identification effort also include updates to the system's overall architecture.

3.2 The Product Evaluation phase

The Product Evaluation phase encompasses the process of creating prototype software for temporary integration and testing of the candidate COTS products (Figure 6). The goal of the Product Evaluation phase is to compare and identify an optimum set of collaborative COTS products for the final integrated system. As a result, this effort needs to include, at a minimum, the following evaluations: (1) functionality of each individual product and product sets, (2) architecture/interoperability of product sets, and (3) performance of each individual product and product sets. The **Evaluation Preparation** stage includes all the activities from the initial acquisition of selected products for evaluation to the design of the overall experiment and the configuration of the evaluation environment.

The **Functionality Evaluation** stage enables actual verification and validation of the overall capabilities of each individual product and product sets. Prior to this effort, our understanding of these products is based primarily on past experience, marketing literature, conference information, and technical discussions with vendors, and product training. The Functionality Evaluation stage can often be performed initially on an individual product basis, thus requiring a minimum level of configuration support. The complete functionality evaluation often can not be implemented prior to the completion of the next stage. The **Architecture/Interoperability Evaluation** stage evaluates the connectivity and architecture of the candidate COTS

products. Specifically, some issues to be addressed during this stage include: (1) the interactions of software components along the identified system critical paths, (2) the extensibility of the overall integrated architecture, and (3) the compliance with required standards by the integrated components. During this stage, the entire candidate product set, and its dependencies, is installed for evaluation. Furthermore, additional software adapters are often required to support their integration. Once they are integrated, the Functionality Evaluation effort can be

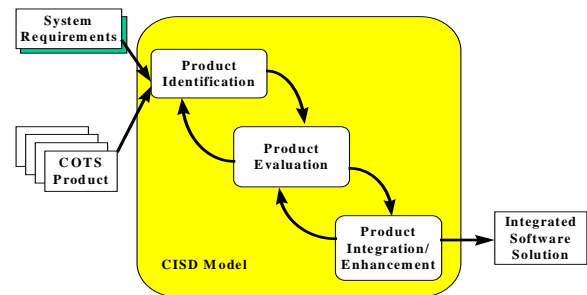


Figure 4 The CISD Model

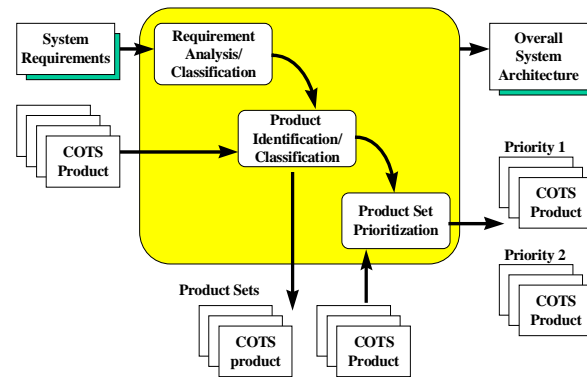


Figure 5 The Product Identification Phase

completed. The **Performance Evaluation** stage is often performed last since the performance criteria are often system-level requirements and can not be easily divided into subsystem-level requirements. If time permits, performance evaluation for each interacting component should also be done to provide detailed understanding of the impact of each individual component to the overall system's performance.

There are multiple approaches to organizing the COTS product evaluation phase. At the two extremes are the Comprehensive Evaluation (CE) and First-Fit Evaluation (FE) approaches. For the CE approach, illustrated in Figure 7, all candidate product sets are evaluated through all identified stages. The result is a prioritized list of these product sets, ranked by their overall performance. The goal of the CE approach is to ensure that an optimal product set will be selected for the final integration at the cost of additional evaluation time and resources. The FE approach, on the other hand, ensures minimal cost to the evaluation effort by eliminating product sets that failed a particular evaluation stage and selecting the first one that passes all evaluation stages. However, the product set selected might not be the optimal solution (Figure 8). Other product evaluation approaches fall between these two extremes.

The result of the Product Evaluation phase is the optimum product set to be used in the final integrated system. In addition, a final System Architecture specification capturing the overall integrated architecture supported by the chosen product set is delivered.

3.3 The Product Integration/Enhancement phase

The Product Integration/Enhancement Phase encompasses all development efforts required to interconnect different selected COTS products into a single integrated system. As described in subsection 3.7, an appropriate development-centric software engineering approach should be utilized to ensure the high quality and reliability of all software to be developed.

4. Integration of development-centric and procurement-centric models

Implementation of large-scale integrated systems often requires the combination of both development-centric and procurement-centric engineering models. Defining a single development approach that incorporates both development-centric and procurement-centric models enables effective handling of projects that require both extensive integration of COTS products and in-house software development. For the DiamondWeb project, the modified staged-delivery (MSD) approach is utilized (Figure 10). As described in [McCO96], the staged-delivery approach integrates the strengths of both Waterfall and Spiral models to support top-down requirement analysis and system architecture definitions, and bottom-up iterative/incremental development at the

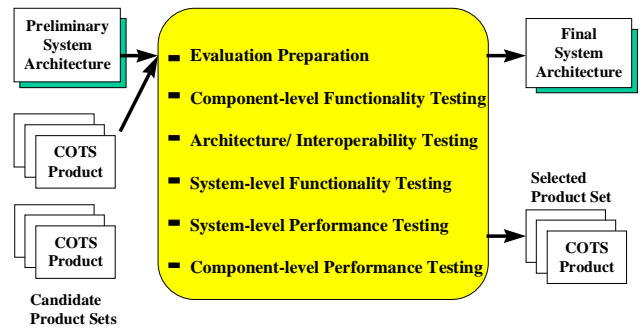


Figure 6. The Product Evaluation phase

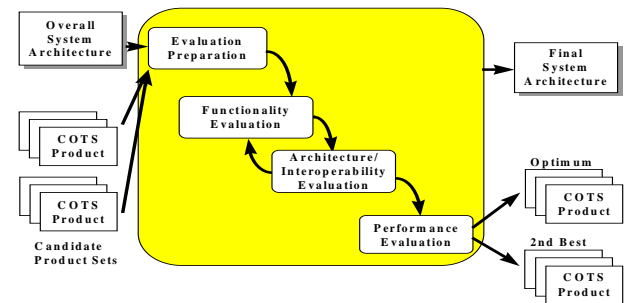


Figure 7. The Comprehensive Evaluation (CE) approach

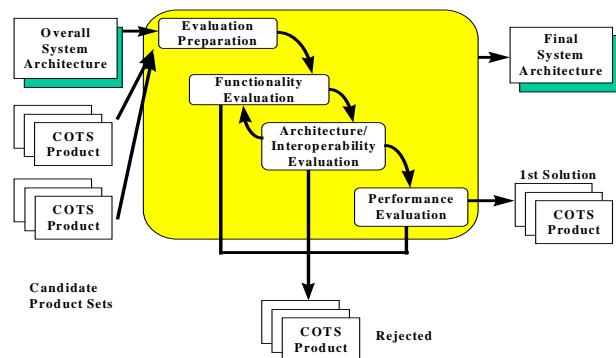


Figure 8. The First-Fit Evaluation (FE) Approach

subsystem level. For the DiamondWeb project, the methodology adopted to support the staged-delivery approach was Octopus [AWAD96], an extension, for real-time system development, of the OMT [RUMB91] and Fusion [COLE93] methods. The MSD includes both staged evaluation/selection of COTS products and staged

software development at the subsystem level. The arrow from the product evaluation stages to the subsystem development stages reflects the adoption of a development-centric approach to implement the additional software needed to support COTS product integration and enhancement. The feed-back arrows from COTS product evaluation stages to System Architecture and Requirement Analysis reflect the dependency of these phases on the COTS product evaluation results. The System Integration and Delivery stages emphasize the incremental and iterative integration of all subsystems, both procurement-based and development-based, into a final system for delivery.

5. Lessons Learned on the DiamondWeb project

The initial result of the DiamondWeb system implementation was encouraging. Despite the multiple CBSE-related technical and management problems that emerged throughout the development effort, appropriate risk anticipation and mitigation have resulted in an on-time delivery of the first version of the product. The following are key lessons learned from this CBSE effort:

1. A risk-based approach to management of requirement changes is needed. The continuing requirement changes during the DiamondWeb development lengthened the product identification and evaluation efforts far beyond what was initially estimated. Overall, these efforts consumed more than half the total development time (starting from requirement analysis). The impact of requirement changes to component-based software development until now has been largely ignored within the research and development community. From our experience with the DiamondWeb project, and many others previously, requirement changes often have a much more serious impact on the overall schedule in CBSE than in other development approaches. This is due to the difficulty with customizing components that are purchased off-the-shelf for integration. As a result, new product identification, evaluation, selection, and integration efforts would be needed to support the new requirements. In the DiamondWeb project, the risk-mitigation strategies that were utilized included: (1) early domain analysis to identify and capture common domain-specific requirements to ensure early establishment of those that are keys to the system, (2) establishment of a requirement change control board to assess the impact of a particular requirement change and determine when the change should take place with minimum impact to the overall development effort, (3) development of alternative

product integration strategies to ensure rapid replacement and deployment of new solutions in response to major changes in requirements, (4) agreements with product vendors to provide custom support for key products such as the real-time operating system, device drivers, Java, and the Web browser, to ensure that these components will evolve to meet the changed system requirements. Although the CISD model used identified a gap between system-level requirements generated during analysis and the subsystem-level requirements needed for early COTS product evaluation, it does not adequately reflect the important relationship between requirement changes and development efforts in CBSE. Efforts are underway to incorporate these informations into the model.

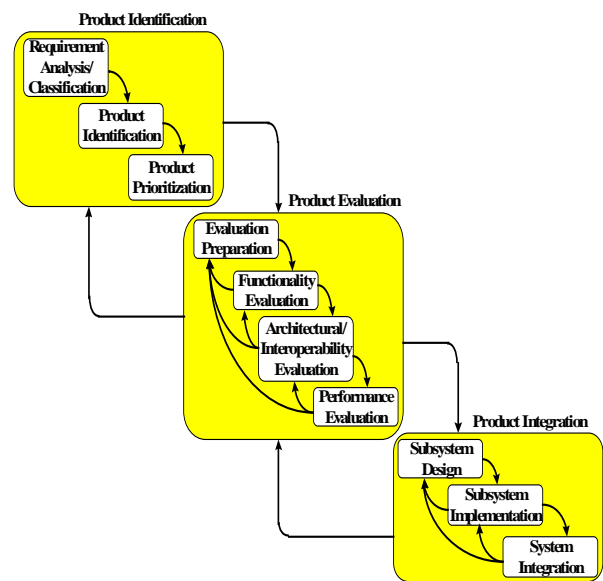


Figure 9 Summary of the CISD model

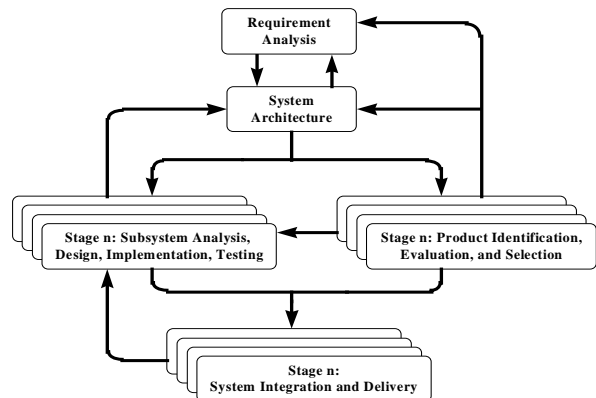


Figure 10 Modified Staged Delivery Development Framework

1. Management of architecture mismatch among different products is needed. The architecture

mismatch among these products was another source of difficulty during our development. Each COTS product selected varied significantly in its functionality and interface making it very difficult to determine the appropriate level of abstraction. Correct abstraction was necessary to ensure that future replacement would not impact the rest of the system. Furthermore, many products selected were unique and could not be replaced when changes occurred later. For instance, during the DiamondWeb development, our replacement of the underlying operating system caused the replacement of several COTS products and in some cases there were no alternative ones available altogether. Additional in-house development had to be undertaken for these irreplaceable products in a much shorter time frame in order to maintain the existing schedule. As a result, the entire engineering staff had to increase its work hours, sometimes to 80 hours/week, to compensate for the shortened allowable development time. As mentioned in previous sections, the issues of architecture mismatch between COTS products are currently being aggressively addressed within both research and development communities.

2. Management of engineering skill mixes for CBSE is important. As COTS products were being replaced during the development, the required technical expertise of the in-house development staff also changed. This change in engineering expertise has caused tremendous management difficulty in the DiamondWeb project. During the beginning of the project, our intention was to allocate all in-house engineering resources at the application development level while relying on COTS products for core services such as the operating system, device drivers and Web browser. The decision to change the underlying operating system during the middle of the development had a tremendous impact on the task assignment of the staff. Since the new operating system came with an sophisticated television control application, there was almost no need for in-house development at the application-level. Instead, because the vendor of this operating system did not provide support for porting it onto some of our specialized hardware devices, in-house development at the device driver level was needed. Since the engineering staff was previously hired and trained for application-level development, retraining was necessary to support device driver development and operating system porting. With the project already under a tremendous schedule constraint, it was very difficult to allocate additional time for the retraining.

6. Summary

The prospect of shorter development schedule, lower resource cost, and higher product quality has led to the increasing adoption of component-based software engineering as a more viable alternative to the traditional development-centric software development approaches. However, underestimating the technical risks associated with integrated system development and management has produced disappointing results. A factor contributing to the current limited success of integrated system development is the absence of a model that explicitly captures the nature and characteristics of this new software engineering approach.

The early adoption of the CISD model in the DiamondWeb project provides a much better understanding of the efforts needed to support its implementation. This is because it explicitly identifies the key efforts required in integrating COTS software components that are often ignored in other development-centric models. In addition, the model integrates well with the iterative and incremental nature of current development-centric approaches.

7. References

- [AWAD96] M. Awad, J. Kuusela, and J. Ziegler, "Object-Oriented Technology for Real-Time Systems: A Practical Approach Using OMT and Fusion," Prentice-Hall, 1996.
- [BROW96] A. W. Brown and K. C. Wallnau, "A Framework for Evaluating Software Technology," IEEE Software, Sept. 1996.
- [CLEM95] P. C. Clements, "From Subroutines to Subsystems: Component-Based Software Development," The American Programmer, Nov. 1995.
- [COLE93] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Haynes, and P. Jeremaes, "Object-Oriented Development - The Fusion Method," Prentice-Hall, 1993.
- [GARL95] D. Garlan, R. Allen, and J. Ockerbloom, "Architectural Mismatch (Why it is hard to build systems out of existing parts)," Proceedings of the 17th International Conference on Software Engineering, April 1995.
- [McCO96] McConnell, S., "Rapid Development: Taming Wild Software Schedules," Microsoft Press, 1996.
- [RUMB91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, "Object-Oriented Modeling and Design," Prentice-Hall, 1991.
- [TRAN97a] V. Tran and D. B. Liu, "A Risk-Mitigating Model for The Development of Reliable and Maintainable Large-Scale Commercial-Off-The-Shelf Integrated Software Systems," Proceedings of the 1997 Annual Reliability and Maintainability Symposium - The International Symposium on Product Quality and Integrity, Jan. 1997.
- [TRAN97b] V. Tran and D. B. Liu, "A Procurement-centric Model for Engineering Component-based Software Systems," Proceedings of the 1997 Fifth International Symposium on Assessment of Software Tools and Technologies, Jun. 1997.