

Defect Proneness forecasting in component based software development: A Generic frame work for OO Systems

Mr. G. Viswanath¹, Mr.N.Bala Krishna²

¹(M.Tech) Computer Science & Engineering, Madanaplle Institute Of Technology & Science,
Madanaplle, Andhra Pradesh, India

²(Asst.Professor, Dept of C.S.E Madanaplle Institute of Technology & Science, Madanaplle, Andhra Pradesh, India)

Abstract: Software systems nowadays are composed from prefabricated commercial components and connectors that offer complex functionality and hold in complex interactions. Unfortunately, because of the diverse assumptions made by developers of these products, fruitfully integrating them into a software system can be difficult often causing budget and plan overruns. A number of integration risks can often be determined by selecting the 'right' set of COTS mechanism and connectors that can be integrated with minimal effort. In this paper we illustrate a framework for selecting COTS software components and connectors ensuring their interoperability in software thorough systems. Our framework is built upon average definitions of both COTS mechanism and connectors and is intended for use by architect and developers through the design phase of a software system. We highlight the function of our framework using a difficult example from the data-intensive systems domain. Our groundwork experience in using the framework indicates an increase in interoperability assessment productivity by 50% and accuracy by 20%.

I. Introduction

The mounting complexity of software systems attached with the decreasing costs of causal hardware has ushered forth the realization of Brook's celebrated "buy versus build" meeting [1]. In the past a production organization spent over a million dollars to build up a customized payroll system over 3 years and another 2 million dollars to preserve and grow it forth rest of its prepared life-cycle. Nowadays still, a business organization cannot afford to spend so much on a customized system that will take over 3 years to realize and a fortune to maintain and evolve. Instead they often opt to purchase a commercial off-the-shelf (COTS) software system (or component) that can accomplish the same desired capabilities. Such COTS systems and components persistently have diminished up-front cost, development time, maintenance, and development costs. These economic considerations often entice organizations to piece together COTS components into a working software system that meets dealing organization's requirements, and the system's functional rations, even at the expense of altering the organization's existing business processes! unluckily over the past ten years numerous studies [2-6] have shown that piecing together available open source and COTS components is quite contradictory from custom development.

Instead of the traditional requirements-design-develop-test-deploy process, COTS-based progress involves activities such as assessment-selection-composition-integration-test-deploy [7-11]. overriding to the success of the entire process, are the assessment and selection of the "right set" of COTS components and connectors. Careful and precise execution of these behavior often ensures the development of a system on time, on resources and in line with the objectives of the project. There are two major mechanism within the assessment and selection process: (1) assessment of COTS functional and non-functional requirements; and(2) assessment of interoperability to ensure that the selected COTS components will suitably interact with each other. While the former has been addressed formerly [7-11] an efficient

solution to the latter has eluded researchers. The first example of such an interoperability issue was accepted by Garlan et al. in [5] when attempting to construct a suite of software architectural modeling tools using a base set of 4 reusable components. Garlan et al. termed this problem architectural divergence and found that it occurs due to (specific) assumptions that a COTS section makes about the structure of the application in which it is to appear that ultimately do not hold true.

The best-known solution to identifying architectural mismatches is prototyping COTS communications, as they would occur in the conceived system. Such a move toward is extremely time-and effort-intensive. The approach compels developers (in the interest of limited resources) to either neglect the interoperability issue overall and hope that it will not create problems during the work and integration phases or it compels them to neglect interoperability until the number of COTS combinations available for selection are cut down to a manageable number (based on functional and quality of service requirements). Both these options add considerable risk to the project. When developers absolutely neglect interoperability measurement they often will be essential to write enormous amounts of glue-code, causing cost and schedule overruns. Otherwise, they risk losing a COTS product arrangement which is easy to integrate, but just "isn't right" because of some low-priority functionality it did not possess. Neither of the above scenario is appealing to development teams.

In addition to the above stated COTS component integration issues, there are issues of utilizing available COTS connectors that occur as well. The study of software architecture [12] tells us that software connectors are the embodiment of the relations and associations between software components. Therefore, ideally, when trying to construct the architecture of a software system, we need to be able to deal not only with the gathering of software components, but additionally the assembly of software connectors. This is exacerbated by the current lack of

considerate in many software system domains (e.g., data-intensive systems [13]) of how to select between dissimilar available COTS connectors. The research literature [14, 15] contains many other studies that describe the enormous difficulty in assembling software connectors by themselves, let alone with COTS software components.

In this paper, we propose an attribute-driven framework that addresses selection of (C)OTS components and connectors to make sure that they can be integrated within project possessions and schedule. One of the key contributions of our work is the classification of connectors to (1) “bridge the gap” between COTS components and ensure interoperability, and (2) satisfy systems quality of service (QoS) supplies. Our proposed construction identifies COTS component incompatibilities and recommends resolution strategies comparatively by using specific connectors and glue-code to integrate these components. Where component exchanges require fulfilling of QoS supplies the framework will propose proper connectors. Such incompatibility in sequence can be used to estimate the effort taken in COTS integration [16], which can then be used as a criterion when selecting COTS products. The framework is non-intrusive, interactive, and tailor able. The quantity conducted by the framework can be approved out as early as the commencement phase, as soon as the development team has known possible architectures and a set of COTS components and connectors. We have tested this framework in a classroom setting and in various example studies, including a challenging real world example from the data-intensive systems domain. Our early experience from using the framework indicates that our come near is feasible, and worthy of active pursuit.

1.1 Definitions

We adopt the SEI COTS-Based System Initiative’s definition [7] of a COTS product: a product that is

- sold, leased, or licensed to the general public;
- offered by a vendor trying to profit from it;
- supported and evolved by the vendor, who retains the logical property rights;
- available in multiple identical copies;
- used without source code modification.

For the purpose of this work we include open-source products as part of the COTS domain except where the source code is personalized by the user (and not redistributed as a fix or a version upgrade). In this paper, we define a component generally as a unit of computation or data store [14]. Components may be as small as a single process or as large as an entire application. Connectors are architectural building blocks used to representation interactions among components and rules that govern those interactions [14]. The rest of this paper is organized as follows. In Section 2, we describe a motivating real-world COTS assessment and selection problem in the data-intensive systems domain. In Section 3 we describe the assessment framework in detail, including the attribute metadata that it captures and how it applies to our example.

In Section 4 we present observed evidence and data taken from a graduate software engineering course at USC that evaluated our framework. Section 5 identifies related works to our own approach and section 6 rounds out the paper with a view of some expectations work.

II. Assessment and Selection Framework

The framework is modeled using three key components, these are: COTS interoperability surveyor, COTS representation attributes, and mixing rules. Inputs to the framework are various COTS component definitions and a high-level system architecture. The output of the framework is an interoperability assessment report which includes three major analyses:

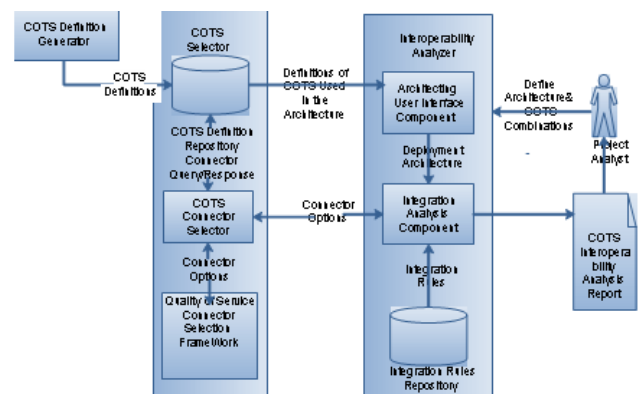


Figure 2. COTS Interoperability evaluation framework

1. Internal postulation mismatches, which are caused due to assumptions made by interacting COTS’ systems about all other’s internal structure [4].
2. Interface (or packaging) mismatches, which occur because of incompatible communication interfaces between two components.
3. Dependency analysis, which ensure that services required by COTS packages used in the system are being provisioned (e.g. Java-based CRM solution requires Java Runtime Engine).

In the remainder of this section we will describe each of the framework components in details.

1.2 COTS interoperability evaluator

To develop the COTS interoperability evaluator we needed to address two considerable challenges:

1. Ensure that the effort spent in COTS interoperability assessment is much less than the effort spent performing the assessment manually.
2. Ensure that the framework is extensible, i.e. so that it can be updated based on popular COTS characteristics.

We tackle these challenges by developing a framework that is modular, automated, and where COTS definitions and assessment criteria can be updated on-the-fly. Our framework allows for an organization to continue a reusable and frequently updated portion (COTS selector) remotely, and a segment which is minimally updated (interoperability analyzer) at client-side. This allows for a dedicated team to maintain definitions for COTS being assessed by the organization. The internal architecture of

the COTS interoperability surveyor component is shown in Figure 2. The architecture consists of the following sub-components. COTS Definition Generator is software utility that allows users as well as COTS vendor to define the COTS components in a generally accepted standard format. Currently we have implemented an XML-based format; however, the implementation format is independent of the primary metadata (e.g., the COTS definition can still be represented using other representation formats, so long as suitable parsers exist). For brevity, we omit its full description of our existing XML format and we point the reader to [17] for a complete description.

COTS Definition Repository is an online storage of assorted COTS definitions indexed and categorized by their roles and functionality they provide (database systems, graphic toolkits etc.). The repository is queried by different sub-components of the interoperability surveyor. In practice, this component would be collective across the organization to enable COTS definitions reuse. Alternately, such a repository could be maintained and updated by a third-party vendor and its access can be licensed out to various organizations.

Architecting User Interface Component provide a graphical user interface for the developers to create the system deployment diagram. The component queries the COTS definition storehouse to obtain the definitions of COTS products being used in the conceived system. Integration Rules storage area specifies various integration rules that will drive the analysis results and interoperability assessment. The rules repository can be maintained

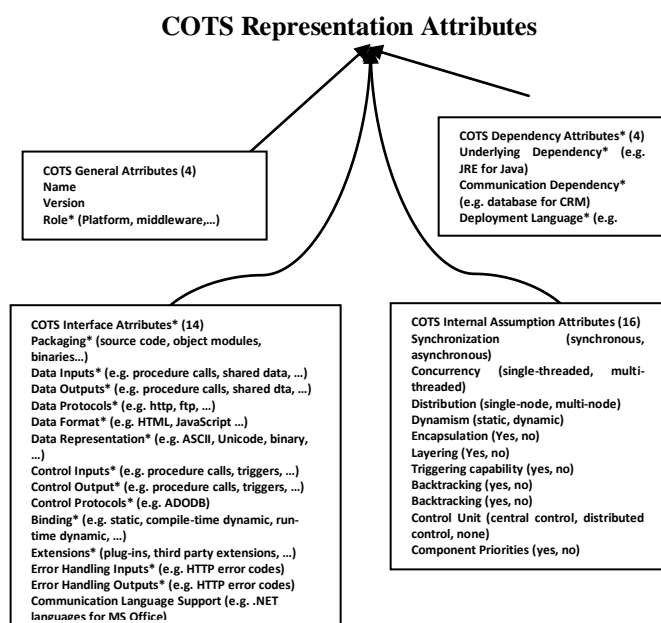


Figure 3. COTS Representation Attributes

remotely; however it will be required to download the complete repository at the client-side (interoperability analyzer) before performing interoperability measurement. This reduces the number of remote queries required when assessing COTS architectures. **Integration Analysis Component** contains the actual algorithm for analyzing the system. It uses the rules specified in the combination rules depository along with the architecture specification to

classify internal assumption mismatches, interface (or packaging) mismatches and dependency analysis. When the integration analysis element encounters an interface variance the component queries the COTS connectors elector component to identify if there is an existing bridge connector which could be used for integration of the components, if not it will recommend in the interoperability analysis report that a wrapper of the suitable type (either communication, or coordination or conversion) be utilized. The integration analysis component then provides some simple textual information (in human readable format) as to the functionality of the wrapping required to enable interaction between the two components. In addition the combination analysis component identifies mismatches caused due to internal assumption made by COTS components, and also identifies COTS component dependency not satisfied by the architecture. For cases where the COTS component definition has misplaced information the integration analysis component will include both an cheerful and a pessimistic outcome. These identifications are both included in the interoperability analysis report. COTS Connector Selector is a query interface used by combination analysis component to identify abridging connector in the event of interface incompatibility, or a QoS specific connector. **Quality of Service Connector Selection Framework** is inextensible component built for identifying quality of service specific connectors. One such extension discussed in this paper aids in the selection of highly distributed and voluminous data connectors. Other quality of service extensions may include connectors for mobile-computing environments that require low memory footprint, or connectors for highly reliable, fault-tolerant systems. To create a quality of service extension, a developer first identifies needed COTS attribute information and ensures the information is captured in the COTS definition repository. This information will typically describe the scenario requirements for COTS connector selection for the particular level of service, e.g., for data intensive systems, it may include the Total Volume, Number of Delivery Intervals and possibly the Number of Users present in the data transfer. The developer then can construct a simple web-based service that accepts the COTS connector definition information, and any other needed data, and then returns the appropriate COTS connectors to select to satisfy the desired level of service scenario. **COTS Interoperability Analysis Report** is output by the selector and contains the result of the analysis in three major sections: (1) internal assumptions mismatch analysis, (2) interface (packaging) mismatch analysis, and (3) dependency analysis. This is the ultimate product of the interoperability framework.

1.3 COTS Representation Attributes

The COTS Representation Attributes are a set of 38 attributes that define COTS produce interoperability characteristics. COTS interoperability characteristics defined using these attributes are utilized by the integration analysis component along with incorporation assessment rules (described in the next section) to carry out interoperability analysis. These attributes have been consequent from the literature, as well as our clarification in various software integration projects.

The two major criteria used for selecting these attributes were:

1. attributes should be able to capture sufficient details on the major sources of COTS product mismatches (interface, internal assumption and dependency mismatches) identified - internal postulation mismatches,
2. attributes should be defined at advanced so that COTS vendors are able to provide feature definitions without revealing confidential product information

To date, we have surveyed about 40 COTS products of which 30 were open source. For the non open source COTS we could identify at least 34 of the 38 attributes from the publicly available information itself. We neglected to include many attributes such as data topology, control structure, and control flow because they were either : too detailed and required accepting of internal designs of COTS products for defining them, or could alternately be represented at a higher level by an already included attribute, or did not provide significant mismatches to warrant us including them. We have classified the attributes that we selected into four groups shown in Figure 3. Attributes (or attribute sets) marked with an asterisk indicate that there may be multiple values for a given attribute (or set) for the given COTS product.

The remainder of this section summarize attribute classifications. The full metaphors of all the attributes can be accessed at [17]. **COTS general attributes (4)** aid in the identification and querying of COTS products. The attribute include name, version, role and type. **COTS interface attributes (14)** define the communications supported by the COTS product. An interaction is defined by the exchange of data or control amongst machinery. COTS products may have multiple interface, in which case it will have multiple interface definition. For example: the Apache Web Server will have one complete interface definition for the web-interface (interaction via HTTP), and another complete definition for server interface (interaction via procedure call). These attributes include packaging (source code modules, object modules, dynamic libraries, etc.), data and control inputs, outputs, protocols etc. When developing the COTS product the developer makes certain assumption about the internal operations of the COTS products. The **COTS internal assumption attributes (16)** confine such internal assumption For example developers of the Apache Web Server assume that the software will contain a central control unit which will regulate the performance of the system. COTS internal statement attributes include synchronization, concurrency, distribution and others. **COTS dependency attributes (4)** define the conveniences required by a COTS product i.e. software the COTS product requires for successful execution. For example any Java-based system requires the Java Runtime Environment (JRE) as a platform. COTS dependency attributes include primary and communication dependencies, deployment language support, and execution language support. An example COTS definition using the attributes specified in Figure 3 is shown in Table 1.

1.4 Integration Assessment Rules

The Integration review Rules are a set of rules used to perform the interoperability analysis. Every rule has a set of pre-conditions, which if true for the given planning and components, identifies an architectural mismatch. For example consider one of the architectural mismatches found by Gacek in [4]:“Data connectors connecting components that are not always active”. For the given mismatch the precondition are: 2 components connected via a data connector (only) and one of the components does not have a central control unit. There are similar rules for performing arts interface, dependency, and internal assumption analysis. Interface analysis discovers if there are commonly shared interfaces between two communicating COTS components, if not it includes recommendation on the type of “glue ware” (or “glue code”) required to integrate the components.

enslavement analysis rules verify if the architecture satisfies all the dependencies that a COTS product require. Finally, for internal assumption we leverage upon the mismatches identified in [4] and add new mismatches based on newly added attributes. Currently, our framework is still underdevelopment and we are in the process of evaluating these rules for behavior such as completeness, correctness, tractability and scalability. We have perform an initial appraisal along two of these evaluation target (correctness and completeness) and our ongoing work involves solidifying means of measuring and evaluating our work.

1.5 Framework Application to Motivating Example

To answer the two questions posed in the motivating example section we apply our framework to analyze the system architecture and COTS combinations. In our example there are 2 major considerations when assessing COTS products and implementation technologies to identify interoperability conflicts:

1. Interoperability conflicts when integrate the digital asset management system with the database.
2. Selection of languages to develop the custom components so as to minimize the development effort by leveraging upon existing support provided by COTS products.

Table 1 Definition of Apache 2.0 using COTS Representation Attributes

COTS General Attributes (4)		
Name	Apache	
Version	2.0	
Role	Platform	
Type	Third-party component	
Interface Attributes (14)	Background Interface	Web Interface
Packing	Executable Program	
Data Inputs	Data access, Procedure Call, Trigger	
Data Outputs	Data access,	

Data Protocols Data Format Data Representation Control Inputs Control Outputs Control Protocols	Procedure Call, Trigger	
		HTTP
	N/A	N/A
	Ascii, Unicode, Binary	Ascii, Unicode, Binary
	Procedure call, Trigger	
	Procedure call, Trigger Spawn	
	None	
Extentions	Supports Extentions	
Binding Error Inputs Error Outputs Communication Language Support	Running Dynamic	Topologically Dynamic
	Log s	HTTP Error Codes
	C, C++	
COTS Internal Assumption Attributes (16)		
Synchronization Concurrency Distribution Dynamism	Asynchronous	
	Multi-threaded	
	Single-node	
	Dynamic	
Encapsulation Layering Triggering Capability Backtracking Control Unit Component Priorities Preemption Reconfiguration Reentrant Response Time Implementation Lang Error Handling Mechanism	Encapsulated	
	None	
	Yes	
	No	
	Central	
	No	
	Yes	
	Offline	
	Yes	
	Bounded	
	C++	
	Notification	
	COTS Dependency Attributes (4)	
Underlying Dependencies	Linux, Windows, (OR)	Unix, Solaris
Communication Dependency	None	
Deployment Language	Binary	
Execution Language Support	CGI	

The project analyst should provide the following information for every interaction in the proposed architecture:

- data and/or control interaction,
- Unidirectional of bidirectional interaction, and which component initiates the interaction,

For the interactions where the large volume data transfer connectors (C1, C2, and C3) are required, the analyst will define attributes specific for that QoS (described further in this section). Assume a scenario where D Space is being assessed as the digital asset management system, and MySQL as the database server. The architecting component user interface will automatically retrieve the definitions for D Space and MySQL and pass the architecture, interaction and definition information to the integration analysis component for assessment. The integration analysis component will apply the rules (from integration rules repository) using COTS attributes and based on the deployment architecture definition to identify:

- Common interfaces supported by MySQL and D Space, bridging connectors and the type of glue code (communication, conversion, synchronization or a combination thereof) required [15].
- Internal assumption mismatches between MySQL and D Space
- Verification that the COTS dependencies have been satisfied in the given architecture.
- Recommended languages for the query manager and data recovery component that will simplify developing glue-code between COTS and custom components.

In the event that the two interacting components (D Space and MySQL) do not share common interfaces, it will recognize (using the COTS connectors selector) a connector that can enable announcement between the two components (JDBC-MySQL driver) and output the fallout in the report. The project analyst can use these findings to estimate the effort required to test (for internal assumption mismatches) and integrate the COTS and custom apparatus [16]. Development

teams can run such evaluation on all their COTS and custom combinations; and use the exertion results as an input to their COTS evaluation table [10] to facilitate the COTS component collection decision. External users, when selecting COTS products and implementation technologies to develop application can run our framework by keeping COTS and technologies selected by JPL and ESA constant and varying their choices of COTS harvest to identify the set which will require minimal integration effort.

To deal with the range of connectors to support large-volume data transfer between JPL and ESA (as shown in Figure 1), and the external users, we employ a specific level of service extension that we have constructed for large-scale, data-intensive systems. The extension was particularly motivated by our experience developing such systems at JPL. A careful, detailed description of this level of service extension is beyond the scope of this paper: for more information on its internal architecture, motivation and objectives the reader is directed to [13]. However for

the purposes of our COTS assessment and selection framework, we will explain the data-intensive level of service extension's high level architecture focusing on two of its critical phases: connector classification, and selection. The inputs to the data-intensive level of service extension are called Distribution Connector Profiles, or DCPs. DCPs capture metadata describing connector Data Access, Distribution and Streaming information abstracted from Mehta et al.'s [15] connector taxonomy.

The profiles also contain information critical to data distribution including Delivery Intervals, Number of Users and Total Volume (in total, there are eight core dimensions of data distributions we are focusing on). To generate DCPs, an architect can manually classify a set of COTS distribution connectors: or the DCPs can be generated using some automated process. For our motivating example, we assume that the production of DCPs has already been performed offline, and we assume the presence of a knowledge base of DCPs resultant from the arrangement. The connectors profiled for the knowledge base take account of connectors C1, C2, and C3 shown in Figure 1. Connector selection starts after the system architecture has been arrived upon and after data allocation scenarios (e.g., constraints on the DCP metadata) have been identified by the user(s) of the system. In our example, there are three distinct distribution scenarios to consider (represented below in human readable form and then following in constraint query format):

- S1. Distribution of data from JPL scientists to ESA scientists
- S2. Distribution of data from ESA scientists to JPL scientists
- S3. Distribution of JPL and ESA data to the outside community

The three scenarios can be expressed as the following constraint queries against the DCP metadata:

```
(Total Volume > 100 GB) ^ (Number Of Users = 2) ^
(S1)
(Num Delivery Intervals = 4) ^ (Volume Per Interval = 25
GB)
^ (Num User Types = 1) ^ (Geographic Distribution =
WAN)
(Total Volume = 1GB) ^ (Number Of Users = 4)
```

For instance, S1 represent the user preference of the

ESA scientists who would like to receive their JPL colleagues' data. S1 describe a allotment scenario in which the producer of data (the JPL scientists) are conveyance over 100 GB of data using a wide-area network (WAN) to 2 regulars of data (the ESA scientists), using 4 escape intervals where each interval consists of 25 GB of data. In S1, from the perspective of the producer of data (JPL scientists), there is a single user type, the ESA scientists. Queries S2 and S3 are formulated similarly. Using S1-S3 as selection criteria, "candidate" connectors are chosen based on their DCP metadata at hand in the DCP knowledge base using a conventional database attribute matching

approaches. After candidate connector filtering, the distribution connectors are assessed for architectural mismatches that may result from their combined use in support of the given circulation scenario.

This appraisal is conducted using an conservatory to Gacek's [4] simple pair-wise mismatch algorithm that compares two architectural elements (in this case, distribution connectors) along the metadata values provided by the DCP. For every value, our algorithm detects potential mismatch areas and decides whether the (set of) mismatches identified are severe enough to avoid connector combination, otherwise, selects suitable connectors that could be used together. For example there may be a divergence in the Number of Users dimension of two of the freedom connectors from the JPL and ESA system, C1 and C2. If C2 supports fewer users than connector C1, then C2 may become the restricted access in the distribution. The detected mismatches are labeled using a simple, but adaptable set of divergence levels, such as severe, or allowable. A severe mismatch may prevent combination of two otherwise matched connectors, while an allowable label may still allow their combination.

The levels are tailor able and meant to be profiled to suit each respective user of our framework. Relating back to our motivating example, the algorithm may decide based on the DCP metadata, that connectors C1 and C2 should be combined, and to combine the Grid FTP [18] connector with an HTTP based custom COTS data connector. Additionally, the level of service extension may conclude that connector C3 can be implemented using an available OTS peer to-peer distribution connector, such as Bit torrent.

III. Empirical Results

In bounce semester of 2006 we conducted an conduct experiment in a graduate software engineering course at USC using our assessment framework. The course focus on software classification development [19] requested by a real-world client. Over the last few years the course has twisted systems for e-services, explore (medicine and software), as well as commercial selling domains. Graduate students enrolled in the track form teams of about 5 members to design and implement a software system within a 24-week time period. During this period the project progresses through inception, elaboration, manufacture and transition phases. Our experiment was conducted close to the end of the elaboration phase, when the team propose a system architecture that would meet the system requirements. We asked 6 teams, whose architectures included at least 3 or more COTS components to use our framework on their personal projects and measured results in four areas:

1. Accuracy of interface incompatibilities notorious by the framework considered as $1 - (\text{number of interface incompatibilities missed by the team} / \text{total number of interface incompatibilities})$. Interface measurement results produced by our framework were verified later through a survey when the teams actually integrated the COTS products. Results in this area price the completeness and correctness of our interface assessment rules.
2. Accuracy of dependency identified by the framework calculated as $1 - (\text{number of dependencies missed by the}$

team / total number of dependencies). Addition assessment results produced by our framework were also later verified through a survey after the project was implemented. These results evaluate the completeness and correctness of our interface dependency rules.

3. Effort spent in assessing the architectures using the framework contrasting to the effort spent in assessing the architectures manually by an comparable team. These results demonstrate the competence of using our framework to perform interoperability assessment as opposed to performing a manual assessment.

4. Effort spent in performing arts the actual integration after using the framework as opposed to effort spent by an equivalent team.

Results here validate the overall utility of our framework Equivalent teams were chosen from past CSCI 577 projects such that they had analogous COTS products, similar architectures, and whose team-members had similar years of experience in project enlargement.

Upon performing autonomous T-test [20] for the metrics above we recorded the results shown in Table2. Our results indicate that the framework increases dependency assessment accuracy and interface assessment accuracy by more than 20% and reduces both assessment effort and integration effort by approximately 50%. These results are significant at the alpha = 5% level.

Table 2. Empirical assessment of our framework

Groups	Mean	Std-Dev.	P-Value
Interface Assessment Accuracy			
Before using the framework	76.9%	14.4	0.0029
After using the framework	100%	0	
Dependency Assessment Accuracy			
Before using the framework	79.3%	17.9	0.017
After using the framework	100%	0	
Effort spent in performing architecture assessment			
Projects using the framework	1.53	1.71	0.053
Equivalent projects	5 hrs	3.46	
Effort spent when integrating the COTS products			
Projects using the framework	9.5	2.17	0.0003
Equivalent projects	18.2	3.37	

IV. Conclusion and Future Work

This paper presents a construction that enables evaluation and selection of COTS workings and connectors early in the software development lifecycle. The skeleton does not eliminate detailed testing and prototyping for evaluating COTS interoperability, however it does provide an analysis of interface compatibilities and dependency. The framework recommends connectors to be used or glue code required and early indications of probable incompatibilities during system integration. Moreover, since the framework is automated it enables evaluation of large number of architectures and COTS combinations, increasing the trade-off space for COTS component and connector selection. Currently we have completed a tool prototype to enable such examination and are in the process of developing a fully functional tool suite. We are also planning experiment to gather empirical data to further test the utility of the framework across a larger sample size and in different development environments. It is also important to note that attribute for frameworks such as ours must be periodically updated based on prevailing COTS characteristics.

References

- [1] F. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering," Computer, vol. 20, pp. 10-19, 1987.
- [2] A. Abd-Allah, "Composing Heterogeneous Software Architectures," Ph.D. Thesis, Univ. Southern California, 1996.
- [3] L. Davis, et al., "The Impact of Component Architectures on Interoperability," J. Systems and Software, 2002.
- [4] C. Gacek, "Detecting Architectural Mismatch During Systems Composition", Ph.D. Thesis, Univ. Southern California, 1998.
- [5] D. Garlan, et al., "Architectural Mismatch or Why it's hard to build systems out of existing parts," In Proc. ICSE, 1995.
- [6] D. Yakimovich, "A Comprehensive Reuse Model for COTS Software Products", Ph.D. Thesis, University of Maryland College Park, 2001.
- [7] C. Albert, et al., "Evolutionary Process for Integrating COTS-Based Systems (EPIC)," CMU/SEI Technical Report CMU/SEI-2002-TR-005, 2002.
- [8] K. Ballurio, et al., "Risk Reduction in COTS Software Selection with BASIS," In Proc. ICCBSS, 2003.
- [9] B. Boehm, et al., "Composable Process Elements for COTS-Based Applications," In Proc. 5th Intl. Workshop on Economics-Driven Software Engineering Research, Oregon, 2003.
- [10] S. Comella-Dorda, et al., "A Process for COTS Software Product Evaluation," In Proc. ICCBSS, Orlando, Florida, 2002.
- [11] Y. Yang, et al., "Value-Based Processes for COTS Based Systems," IEEE Software, vol. 22, pp. 54-62, 2005.
- [12] M. Shaw, et al., Software architecture: perspectives on an emerging discipline. Upper Saddle River, N.J.: Prentice Hall, 1996.

- [13] C. Mattmann, "Software Connectors for Highly Distributed and Voluminous Data-intensive Systems," In Proc. ASE, Tokyo, Japan, 2006.
- [14] N. Medvidovic, et al., "A Classification and Comparison Framework for Software Architecture Description Languages," IEEE TSE, vol. 26, pp. 70-93, 2000.
- [15] N. Mehta, et al., "Towards a Taxonomy of Software Connectors," In Proc. ICSE, Limerick, Ireland, 2000.
- [16] C. Abts, "Extending the COCOMO II Software Cost Model to Estimate Effort and Schedule for Software Systems Using Commercial-Off-The-Shelf (COTS) Software Components: The COCOTS Model", Ph.D. Thesis, Univ. Southern California, 2004.
- [17] J. Bhuta, "A Framework for Intelligent Assessment and Resolution of Commercial Off-The-Shelf (COTS)Product Incompatibilities," USC, Tech. Report USCCSE-2006-608, 2006.
- [18] C. Kesselman, et al., "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," Intl' Journal of Supercomputing Applications, pp. 1-25, 2001.
- [19] B. Boehm, et al., "A stakeholder win-win approach to software engineering education," Annals of Software Engineering . vol. 6, pp. 295 - 321, 1998.
- [20] T. Dietterich, "Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms," Neural Computation, vol. 10, 1998.
- [21] E. Mancebo, et al., "A Strategy for Selecting Multiple Components," In Proc. ACM SAC, Santa Fe, NM, 2005.
- [22] L. Davis, et al., "A Notation for Problematic Architectural Interactions," In Proc. ESEC/FSE, 2001.
- [23] R. Keshav, et al., "Towards a Taxonomy of Architecture Integration Strategies," In Proc. ISAW, 1998.

BIOGRAPHIES



Engineering College, Kankamma Chatram, Chittoor District, India.

G.Viswanath is born in 1982 in India. He is graduated in **B.C.A** from Osmania University, Hyderabad and post graduated in **M.C.A** from S.K. University, Anantapur. Presently he is doing post graduation in **M.Tech (C.S.E)** at MITS College, Madanapalli, Andhra Pradesh, India from JNTU Anantapur. At present working on Assistant professor at Sai Sakthi



N. Bala Krishna is born in 1984 in India. He is graduated in **B.Tech** from JNTU Hyderabad. He is post graduated in **M.Tech** from JNTU Anantapur. He is currently working as a Assistant professor in the department of Computer science and engineering at MITS College, Madanapalli, Chittoor District, Andhra Pradesh, India.