

# Discovery of Best-Fit UI-Component using Comprehensive Analysis

A.V.Sriharsha<sup>1</sup>, Dr. A. Rama Mohan Reddy<sup>2</sup>

<sup>1</sup>Sree Vidyanikethan Engineering College, Tirupati

Email: avsriharsha@yahoo.com

<sup>2</sup>Sri Venkateswara University, Tirupati.

Email: ramamohansvu@yahoo.com

**Abstract**—Software Engineering provides low level abstraction of application development framework. Popular software engineering paradigms are chosen for development of many large applications. In the course of maintenance of an application, there are famish needs of reengineering of the application and configuration of improving domain based functionality, contextually leads to, a grim necessity for adding a new pieces of code or component or functional unit to the application. Identifying the matching UI component and tuning to the existing large applications is a challenging issue which is called “composition of software components”. The high level design and architectural concepts those are available for composition and integration of large applications are already in vogue provided by CASE tools and Software Architecture analysis and design techniques (Garlan and Mary Shaw). In this paper, we propose a framework that applies comprehensive analysis and data mining for the selection of parameters and subsequently fitting best UI component. This work also focuses how statistical correspondence analysis is useful for analyzing the UI configuration data.

**Index Terms**— software architecture, user interface design, configuration management, knowledge engineering.

## I. INTRODUCTION

Studies have shown that somewhere between 15% and 40% of large software projects fail. The larger the project, the more likely it is to fail. So many books have been written about some these monumental software failures. There are many reasons for these failures. These numbers are appalling.

### *Monolithic Programs*

Monolithic programs are asking for a software project failure. To be more precise, One should say large monolithic programs are asking for a software project failure. In Visual Basic terms, a project built from a single Standard EXE is a monolithic program. A monolithic program has these flaws:

- Totally unrelated code coexists in the same EXE file.
- Any change to any part of the code requires recompiling and re-releasing the entire program.
- Any code throughout the project may access any form, class modules, public procedure, or public variable resulting in spaghetti code.
- The code is difficult to reuse at design time.

- The code is impossible to reuse at runtime, except through the "shell" function.
- No person on the programming team can get a handle on all the code.

Visual Programming Workbench facilitates reuse of code at design time by including the same form/container, class module, or code module into multiple projects, but you must be very careful that any change to the shared code does not break any of the other projects. As the code increases, the project becomes entirely unmaintainable. For small programs, remembering is easy. A human professional can probably remember 100 things even on a hard trial, but it is very difficult to remember 10,000 things, much less a million. When the situation is brought to that stage, there is a rule of programming:

*The day you write the code, you and God knows what it does. A week later, only God knows.*

Most of the large software projects that have failed (including Omega) were written as monolithic programs.

### *Program Quality*

Bertrand Meyer, in his book Object-Oriented Software Construction, listed these factors to identify the quality of a program: [3]

- Correctness: The ability of software products to exactly perform their tasks, as defined by the requirements and specifications
- Robustness: The ability of the software systems to function even in abnormal cases
- Extendibility: The ease with which software products may be adapted to changes of specifications
- Reusability: The ability of software products to be reused, in whole or in part, for new applications
- Compatibility: The ease with which software products may be combined with others
- Efficiency: The good use of hardware resources, such as processors, internal and external memories, communications devices
- Portability: The ease with which software products may be transferred to various hardware and software environments

- **Verifiability:** The ease of preparing acceptance procedures
- **Integrity:** The ability of software systems to protect their various components against unauthorized access and modification
- **Ease of Use:** The ease of learning how to use software systems, operating them, preparing input data, interpreting results, recovering from usage errors.

#### *Defining Components*

The concept of building parts to a predefined specification has been around for more than a century. Today, the manufacturing sector can produce many devices in large quantities using interchangeable parts to assemble products. CBD applies the same principles to software development. Parts designed according to predefined specifications can be assembled to create applications. These parts are known as components. Software components are “binary units of independent production, acquisition, and deployment that interact to form a functioning system.” The goal of component-based software development is to standardize the interfaces between software components so they can work together seamlessly. Encapsulated, Replaceable, and Independent Components are considered to be black boxes.

#### *UI Components*

The user interface components are essential for any software application. There are a range of technologies that can be used to integrate Web-based user interfaces into software applications. The availability and adoption of different technologies results in different integration architectures with different system characteristics. For the relevant stakeholders to make informed decisions that are particularly suited to their application context, it is important that the available integration architectures, their characteristics and their relationships to the various technologies are identified.

#### *Key User Values*

1. **Mobility:** People want to send and receive voice calls and messages anywhere/anytime.
2. **Ease of use:** People seek simplicity despite underlying complexity.
3. **Controlled accessibility:** People want control over who gets through to them, when they can be reached, and by what means (voice, voice message, note, etc.)
4. **Personalization:** People want to customize the interface to their unique requirements
5. **Call closure through multitasking:** People to complete a communication task using either voice or note communication, or both simultaneously.

The design concepts and the list of preliminary user values represent the design team’s hypothesis of what will make a successful product. This hypothesis is later tested in the Refinement and Analysis Stage.

#### *Limitations in Task Analysis*

*Limitations of Task Analysis for New-Generation Product Concepts:* For new-generation products one of the limitations of the task-analysis process is that there is a large gap between the existing work model (i.e., work carried out with current products) and the enhanced work model (i.e., work carried out with the new design concepts). New-generation products encompass features and services for tasks that do not yet exist, and detailed task analysis of the enhanced model would constrain the creativity of the design team without contributing useful information. In the analysis Stage of new-generation products, the focus is more on discovering and verifying user values and detailed product attributes required to deliver these values. The success of this strategy relies upon an iterative design process with substantial user input.

#### *Software Quality Attributes*

General software quality attributes include scalability, security, performance and reliability, and their requirements are part of an application’s nonfunctional requirements, which capture the many facets of *how* the functional requirements of an application are achieved. To be precise, specific quality attribute requirements are requisites for design of UI component of an application.

## II. PROPOSED WORK

#### *Basic Characteristics of Component:*

**Granularity: ( $B_g$ )** The level of using the component based on its application of the function in the host application. The component can be allowed to singly create its instance or multiply. The basic characteristic of granularity determines the number of installed or used instances (*physical instances*) of the component.

Component may be deployed in the application exhibiting various polymorphic features. This also calculates to the increasing size of the host application.

**Metadata: ( $B_m$ )** The support information that is given to the component to adjust and adapt itself in the host application in order to serve the intended functionality.

**Connectivity: ( $B_c$ )** The protocol information and the structure of data that is transferred from the component and the application. More to emphasize is the communication media between the component and the application.

**Functional Coupling: ( $B_f$ )** The information communication strength between the host application and the component. The amount of data communicated between the host application and the component.

**Intra-Module Coupling: ( $B_{imc}$ )** The component not necessarily be an atomic component, it may be composed by various types of sub-components. The information

transaction or communication between the sub-components within the component category border is latently quoted as intra-module coupling.

Apart from the above characteristics Historical Awareness deals with heterogeneous artifacts of the components. The heterogeneity of the component can be determined by means of calculating unmatched functional specification of the component with respect to the host application.

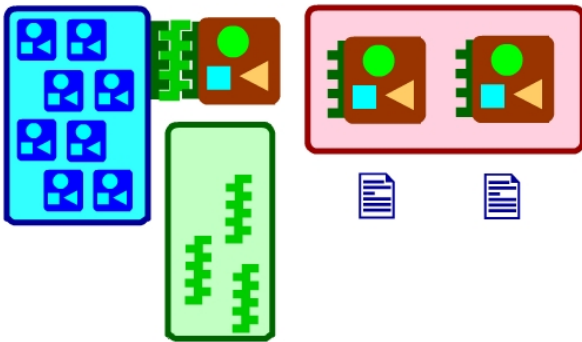


Figure 1. Illustrating the selection of best fit glue configuration to adapt the external component into the host application.

### THEORETICAL ABSTRACTION

$B_g$	basic-granularity
$B_m$	metadata
$B_{fc}$	basic-functional-coupling
$B_{imc}$	basic-intra-module-coupling
$R_c$	pre-installation-stage-loc
$P_m$	post-installation-no-of-modules

The above mentioned characteristics of a component present factual details which are observed throughout the stages of the component life cycle with respect to the host application. The direction of the underlying work is to determine the inference from the various probabilistic relationships between various types of characteristics of the component. The probabilistic relationships between the characteristics can be quantitatively determined using the log and the configuration files of the component. The configuration file of the component determines the implicit best fit state of the component. The probabilistic relationship is, otherwise various fitting configurations of the component. The configurations of the components are made available for all the stages of the component development. This forms a very huge textual repository of the configurations, the huge semi-structured text database. The huge repository of the all available configurations is processed by a text mining utility [7][8].

### III. CONCLUSION

Components are very essential building blocks for any applications, especially for the large applications which are derived from lots of legacy code (components). Selection of a component from the external or third-party

vendor to the application to enhance the functional specification of the software is to be done scrupulously.

Data mining theories are applied for the selection of best fit component. Various algorithms can be developed to adopt the component into large software architecture. Future research can be progressive in developing various types of Architecture Re-Engineering Tools (ARE) tools.

User Interface Components are very essential for all types of software applications. Selection of user interface components is more crucial, for a rapid application development workbench. Such comprehensive analysis working on various types of non-functional attributes is very important, which paves the conceptual selection of the UI component for the applications.

### IV. REFERENCES

- [1] <http://www.codingthearchitecture.com>
- [2] <http://www.softwarearchitecturenotes.com>
- [3] Object-Oriented Software Construction, Second Edition by *Bertrand Meyer*; Prentice Hall PTR, 1997, ISBN 0136291554, 2nd edition.
- [4] Software Component Integration Testing: A Survey, Mohammed Jaffarur Rahman, Fakhra Jabeen, Centre of Software Dependability, Mohammed Ali Jinnah University, Pakistan. 2006.
- [5] Data Mining: Concepts, Models, Methods, and Algorithms by Mehmed Kantardzic, John Wiley & Sons 2003.
- [6] Data Mining: Concepts and Techniques, Second Edition, Jiawei Han, University of Illinois at Urbana-Champaign, Micheline Kamber.
- [7] Beyond Software Architecture: Creating and Sustaining Winning Solutions, Luke Hohmann, Addison Wesley, January 30, 2003, ISBN - 0-201-77594-8.
- [8] Model-Driven Architecture in Practice, Oscar Pastor, Juan Carlos Molina, Chapter 19, Springer-Verlag Berlin Heidelberg - 2007.
- [9] "Component-Level Comprehension in Large Applications using Data Mining Theories", A.V.Sriharsha, Dr. A.Rama Mohan Reddy, Proceedings of International Conference on Systemics, Cybernetics and Informatics, Pentagon Research Centre, ICSCI 2009.