Software Engineering Design Issues

[1]Soma Akhil, Soma Anvesh[2]

[1]*Accenture, India*, [2]*TCS, India*

G-15, Krishna Arcade, Nizampet, Kukatpally, Hyderabad, Telangana, India-500090

***Abstract*— Research in software engineering is concerned with the automation and the enhancement of the processes of building some computer related application systems. In this paper, we mentioned some significant software engineering problems from the context of developing very large information systems. The aim of the paper is to investigate applicability of object-oriented software design patterns in the context of aspect-oriented design. The main assumption is that there exist design patterns that solve software engineering paradigm independent design problems and that such patterns, in the contrast to the patterns solving paradigm-specific design problems, can be expressed in terms of any software engineering paradigm.**

Keywords- VLSI View point, Software Development issues, System evolution, Paradigm problem, Large scale Integration, GoF23 patterns.

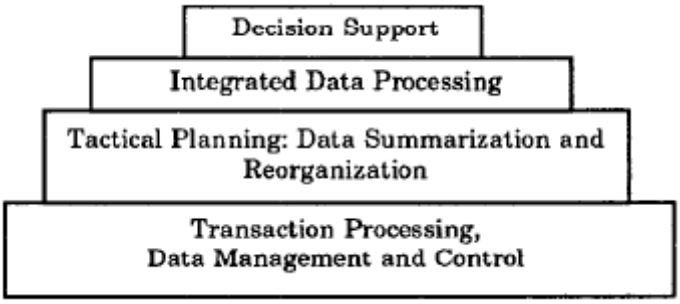
I. INTRODUCTION

The ultimate aim of research in software engineering (SE) is to improve the quality and productivity associated with both the products and processes of software development. The foreseen benefits are improvements in the management of the development process, increases in user satisfaction, and the delivery of greater functionality to the market place in the form of software and related products. A number of software engineering paradigms, approaches and methodologies exist today. Although the object-oriented (OO) paradigm still remains one of most popular, it is gradually replaced by the aspect-oriented (AO) because of the concern crosscutting problem.

Some of the major problems associated with the automation of software development occur with respect to requirements specification, design, reusability, maintenance, validation, verification, and testing. The approach to research on the problems of software engineering is founded on a different class of application domains; namely that of very large information systems (VLIS). In the following sections we briefly define and characterize VLIS and the problems associated with VLIS development. The intent is to contrast the nature of the software development problems. As a result we hope to illuminate new perspectives on the software engineering problem and construct a foundation from which an integrated, productive research program can be launched for software engineering issues in VLIS.

II. VLSI View point

VLIS are federations of subsystems developed according to a system wide design plan to provide information to support the operational, decision-making, managerial and analytical functions of an organization.



Information Systems Pyramid

The operational scope of the VLIS across different levels can be characterized as a pyramid structure in business activities. As if we consider the case with embedded systems, VLIS development is mainly concerned with the performance of software and the functional complexity. In essence, the complexity of a VLIS is rooted in the demographic complexity of the application environment and the size of the system. SE research therefore needs to contain a strong empirical component and should be grounded by data generated from the industrial experiences. The formulation of research problems and the strengths and weaknesses of the alternative approaches need to be subjected to the test of time constraints, resource limitations, industrial practices and economic pressures.

III. Software Development Issues

This mainly describes some of the issues associated with the software development process in the context of VLIS. Here in each case we define the issues, find the sources of its existence, and discuss the implications of what to do accordingly. And then we see these issues as highly interdependent with other and regard the approach to an overall solution as necessarily dependent on concurrent development in each and every area.

A. System Evolution

Currently, the process of maintenance is too often similar to that of patching of software bugs. The software maintainability is assumed wrongly to be a natural byproduct of good systems development. These perceptions are faulty and thus results to the maintenance crisis being experienced today. So today in order to support the scope of maintenance, it must be recognized that software systems need to evolve and that this evolution of systems cannot be treated in the same manner as initial development.

The ability to continuously evolve is crucial for VLIS. Information systems are typically a critical arm for the business and

it must adapt continuously to the changing business environment. Change is not the exception for these software systems, it is just the norm. Unless they can effectively adapt to the needs of the company, the competitive position of the company and the stability will be jeopardized.

So in order to support system evolution, information required during maintenance must be available and processes which directly support maintenance must be developed. Taken together, design rationale and the maintenance processes must help the designer (1) understand the structure, functionality and behavior of a system (2) assess the level of impact of alternative changes, and finally (3) control of the impact of necessary changes that are made. Of course, maintainability and the integrity of the system must be preserved.

Supporting system evolution must be viewed as a major factor in shaping an approach to formalizing the design process, Maintenance requires more information because an existing system represents a large set of constraints which does not exist during initial design and the design tradeoffs made during maintenance differ from those in the initial design process. For example, data structure decisions maybe driven by efficiency and quality during initial development but by minimization of impact during maintenance.

B. Leveraging Expertise and Experience

Necessarily large size of VLIS development team members combine with the relative scarcity of highly experienced builders mandate the judicious use of less experienced personnel in most of the phases of development. Leveraging expertise and experience is concerned with the ability to successfully complete and manage large software development projects with a relatively small percent of highly experienced and proven resources. This is the critical issue given that the success of a project development is directly correlated with the levels of experience and talent across members of the project team.

The main focus of SE research should be on supporting the high level design activities where the dividing of high level problems into relatively independent components takes place. The simple functional demands of individual programs allows for their construction by less experienced members.

C. The Design Process

The design process is the task of generating or creating design byproducts and subsequently refining, evaluating, integrating, and modifying these byproducts until the final result satisfies the initial requirements that are taken according to the problem definition. In essence, the design process is the task of mapping problem requirements to design solutions. The design process should be guided by a economic, productive and controllable methodology that will ensure a high quality product.

The reasons why the design problem is important are necessary. Good design decisions made early have a positive effect on the efficiency of the development process and the quality of the ultimate product. Poor design decisions declines the efficiency, quality and cost of the development process and the design products.

Design of the software development is categorized by a necessity to deal simultaneously with a large number of diverse constraints. In general the design problems are intractable. However, we believe that the combination of the restricted domain of VLIS development, the knowledge accumulated from the experience of building a large number of VLIS and the simplicity of its algorithmic requirements provides good insights into the available expertise, known alternatives and solutions. Mainly our intention is to exploit the natural structure in ways that allow us to reduce the complexity of the problem to manageable levels. We mention several issues that help focus some of the design process concerns.

1) The Paradigm Problem:

The paradigm problem mentions to the failure to develop and recognize a productive, manageable, economically feasible process model for SE. Attention has been focused on the development of new software engineering paradigms, but till now no results have proven completely satisfactory for VLIS development. Any successful model must deal with the interdependent facts of making design decisions while recognizing the need for adequate leverage and project management.

2) Bridging the Functional to Technical Gap:

A notable portion of the design process occurs when translating the business problem description in to a high level systems design. Today, the techniques at this level of the development process are not clearly understood. The result is we cannot able to map problem requirements to technical solutions. No good languages have been developed in which the requirements of the problem can be expressed and ultimately transformed to technical solutions.

3) Design Evaluation:

So in order to make good design techniques and decisions, one must have the ability to assess the quality and validity of a particular design decision or can able to weigh the relative merits of competing design alternatives. The lack of evaluation ability definitely leads to inadequacies in assessing the impacts of a design decision on all factors of the design process. Under the umbrella title of evaluation we include testing, verification, validation and, as a specific instance, prototyping.

4) The Representation Problem:

The representation problem is a elementary requirement for advancement in each of the areas mentioned above. The issue is the ability to manipulate, express and make inferences about design decisions and processes. Currently, major development takes place at very low level design for at least two reasons. Firstly, current methods of software engineering encourages the designers to think in terms of low level issues such as data structures, performance measures, database, screens, and interfaces etc because these representations are the only mechanisms that provide evaluation feedback and feasibility measures. So as a result designers move to lower levels without adequately investigating alternative early design decisions. Second, the nature of the business is that cost pressures often do not allow for a need to investigate on high level design

alternatives. As a result, projects designs are often inadequate and easily damaged.

5) Large Scale Integration:

The major problems of large scale integration is concerned with the understanding, usage and exploitation of the functionalities, protocols, standards and communication interfaces of a heterogeneous set of technologies in developing large systems. Integration of different component systems differing both in functionality and platform is important because systems within the commercial environment can no longer be treated as isolated entities. In fact there is great discourage to do so.

The large scale integration problem mainly points out the need to understand the interrelationships of all systems within a company. Efforts need to be concentrated on large scale design at the enterprise, or companywide level before detail design and implementation of a particular component is undertaken. Understanding the enterprise level connectivity and integration issues is extremely important and will have tremendous impact upon the design process.

IV. AO Solutions of Paradigm Independent Design Problems

If GoF23 pattern can be applied in AspectJ by using AO only, it can be viewed as a pattern that, respecting for OO and AOP paradigms, solves a paradigm-independent design problem. Despite of it, in such a case, both OO and AO patterns solve the same design problem, their usage differs. The OO patterns solves this problem for objects, whereas the AO patterns solves it for aspects. Let us briefly consider the suggested methodology, to rewrite paradigm-independent GoF23 design patterns for aspects.

- If a GoF23 pattern, maybe with a reduced significance, can be implemented using only singletons, this pattern is considered as a candidate to be a paradigm independent pattern for rewriting in AspectJ.
- All the classes in the candidate pattern should be replaced with aspects and all object constructors should be replaced by the AspectJ static method *aspectOf*, which allows us to access the cite of the aspect.
- The candidate pattern should be analyzed in order to discover and remove irrelevant data members and methods. Some data members and methods can become irrelevant because the aspects which replaced the classes are singletons and because of transformation of some pattern members to fit the point cut model in the pattern.

A. Investigation of the Applicability of GoF23 Patterns to Design the Aspects

Firstly, let us discuss these GoF23 patterns – Singleton, Prototype, and Composite – that are pointless in the aspect-oriented paradigm. The Singleton pattern becomes trivial after rewriting it in AspectJ and disappears. The essence of Prototype pattern is the ability of objects to clone its instances (i.e., create several instances of the same class based on already existing instance). However, in AOP no one needs to clone the aspects. Even if it is possible to use several instances of aspects per object or per control flow, it is not possible to control instantiation in the way to support cloning. Thus, Singleton and Prototype design patterns are senseless in AO paradigm. Senseless is also the Composite pattern because, in the case of OO paradigm, its implementation requires to hold the references from one to another instance of Composite object. In the case of AO paradigm, the solution results in an eternal loop when only one container aspect is defined and this aspect is referenced in a tree at least two times.

We use UML class diagrams to model both OO and AO patterns. To represent aspects in UML models we use stereotypes: Aspect, Advice, Pointcut, and Joinpoint. The latter one represents the relation between the pointcut, described in the aspect, and its actual joinpoints in classes. While modelling the AO patterns by UML, we use the traditional UML relations such as inheritance, association, and dependency. For a better understanding of the diagrams.

V. Conclusions

The intent of this paper is to discuss the suggestions that the process of developing very large information systems (VLIS) has on the approach to the software engineering problem. Although the same SE problems are found in many domains, they take on a unique set of constraints when considered in the context of developing VLIS. It is our position that possibilities for enhancing the software development process are functions of the domain in which one participates. The paper proposes a classification of the ways of solving design problems using OO and AO design patterns. The proposed classification contributes to the better understanding of relations among the design problems and the design patterns. The paper proposes also a technique for redesigning object-oriented patterns into pure aspect-oriented ones and demonstrates application of this technique for the GoF23 design patterns.

References

[1] Craig Gaskell and Armstrong A. *Takang,Professional issues in software engineering the werswective-of UK academics*

[2] Zilvinas VAIRA, Albertas C APLINSKAS *Vilnius University Institute of Mathematics and Informatics Software Engineering Paradigm Independent Design Problems, GoF 23 Design Patterns, and Aspect Design*

[3] David NotkinAutumn *Quarter 2008 Software engineering issues*

[4] RICHARD H. THAYER, ARTHUR B. PYSTER, AND ROGER C. WOOD, *Major Issues in Software Engineering Project Management.*

[5] **Jochen L. Leidner** School of Informatics, University of Edinburgh, *Current Issues in Software Engineering for Natural Language Processing.*