Confronting Antagonistic Views of Software Design

Sergio Martins Fernandes
José Eduardo Belix
Selma Shin Shimuzu Melnikoff
Edison Spina

The Department of Computing and Digital Systems Engineering (PCS)
University of São Paulo - Polytechnic School
Av Professor Luciano Gualberto, travessa 3, nº158 - Sala C2-42.
Cidade Universitária - São Paulo - SP
CEP: 05508-900
Brazil

Abstract: - Despite some skepticism from the software industry about software design, the traditional software engineering approach has been enriched in the last years by new concepts, tools and techniques, like design patterns, software architecture, the UML, frameworks and, more recently, by the emergency of OMG's MDD/MDA. On the other hand, different approaches have emerged, notably the agile processes, opposing many established software design best practices.
This paper briefly presents the approach regarding these software design views, and characterizes their strengths and weaknesses for specific types of software systems developments.

Key-Words: - Software Design, Software Modeling, RUP, Agile Processes, MDD/MDA

1 Introduction

The design of a software system is a key aspect of software development. The software engineering community has always emphasized the quality of software design as fundamental for the effective success of software development projects.
On the other hand, a large part of the software development industry has resisted putting a significant effort on the design of software systems. Even complex systems were built by the direct translation of requirements into source code. The central argument of the industry to justify these practices: tight schedules. Others are the difficulty to keep the model and the code synchronized during the software life cycle; and the developer's lack of skills to use modeling languages effectively.
One answer of the software engineering community to that point of view is to underline the frequent quality and maintenance problems in software systems that are built without a graphic model. Modeling reduces the coding effort and increases quality, making maintenance easier.
Nevertheless, the software engineering community recognizes that the problems appointed by the industry are relevant, and is trying to solve them. The usual path they point involves adequate developers training and extensive use of software tools (like CASE tools) to support the modeling effort and to help automatically synchronize model and code.
The OMG (Object Management Group) adopted the vision of tools supporting modeling, and is developing an initiative named MDA/MDD (model driven development / model driven architecture). MDD establishes standard ways for the elaboration of UML software models that are platform independent, and the automatic translation of that models into other models – platform dependent models – and, subsequently, into source code.
A group of the software engineering community self named agile community followed a different path, questioning the conventional practice of software design. This community advocates what is called evolutionary design, which reduces the importance of a graphic design, accepting it could be directly expressed into the source code.
This paper briefly analyses each of these views, and characterizes their strengths and weaknesses for specific types of software systems developments

2 The Software Industry View

The software industry recognizes the problems concerning software development and maintenance, but it frequently has been skeptical about the

effectiveness of the conventional software engineering best practices to solve these problems.
The software industry acknowledges, generally without much conviction, that software modeling is useful, but argues that there is not enough time to do it, due to the tight schedules of software projects.
Some modeling efforts which are successful at short term, frequently are lost at long term, because it is difficult do keep model and code synchronized, while the software evolves over its life cycle.
This skepticism is partially due to the failure of many software development process implementation projects; and to the lack of knowledge of the software engineering evolution through the last decades. There is also a perception that the software community is adequately skilled to write code, not to build models [1].
The quality improvement and the quantity increase of Computer Sciences courses worldwide are weakening this argument. Nowadays, there is a new generation of software engineers effectively using design patterns or software architecture modeling, although it is less frequent that a comprehensive modeling effort be made.
Researches show that the software industry has a relatively low projects success rate, and insufficient quality records. Still, the growing sophistication of software systems and the spread of software in all human activities and places make it impossible to conclude that the software industry has failed.

3 The traditional Software Engineering Community view

A fundamental software engineering practice is that the graphical, or visual, design should precede coding. Both the structured analysis and object-oriented paradigms incorporate this practice.
The development of the UML, the explicit modeling of the software architecture and the proposition of design patterns have enriched the traditional vision of software design, helping increasing the conceptual knowledge and the ability of software designers to create higher quality software designs, achieving higher quality software systems.
This view is expressed, for instance, in the Rational Unified Process (RUP) [RATIONAL 2003], which proposes an iterative and incremental software life cycle. The initial phases of the RUP life cycle are focused on the definition of software architecture, and on the construction of an executable prototype that validates the architecture. RUP emphasizes the use of design patterns and the definition of what it calls mechanisms (like persistence or messaging mechanisms) for software design. Its Analysis and Design discipline recommends the development of two models: an analysis model, relatively free of technological considerations, which is translated in a design model. Both models describe all the functionality of the software system.
Although it should be configured for specific projects, RUP is, in essence, a relatively formal and prescriptive process. Activities, disciplines, artifacts, roles of the process are very clearly defined and logically organized in workflows [5], [6]. Such level of formalism and the great quantity of artifacts it encompasses makes RUP critics call it *heavyweight.*

3.1 MDD / MDA

Probably the most advanced stage for software design of the conventional view of the software engineering community is the MDD, which is a proposition of the OMG for software development [12].
Its central idea is the elaboration of UML models and the use of specialized CASE tools to automatically translate these models into code.
MDD is based in a standard being developed by the OMG, named MDA, which defines a general architecture and technologies that are the foundation of MDD. Although this effort has already produced results, it is still a work in progress.
It is possible to characterize MDD as the next step for software development, after the third generation programming languages, which use compilers to automatically translate source code into binary code.
The MDD/MDA vision is compatible and develops Jacobson's vision [4]. Jacobson foresees an environment in which the team can focus on the creative tasks of software development – related to business logic definition. Very specialized software tools would supply the other tasks, defined by Jacobson as non-creative.

4 The agile community view

For the last years, a group of the software engineering community gathered to provide a new approach for software development, named light, or agile. They developed many software development processes. The most popular is the Extreme Programming (XP) [2]. This community created a non-governmental organization called The Agile Alliance (www.agilealliance.com), with aims to develop and disseminate their ideas.
The agile processes do not intend to be a recipe for software of any size and complexity. It is focused on small to medium duration software projects, with teams of 10 people maximum, physically close to each other while working on the project.

These processes oppose the conventional approach for software development in many ways. They are not very prescriptive, are highly iterative, and are adept of low level of ceremony: demand very little documentation and formalism [5].
Regarding software design (the focus of this paper), the agile processes do not emphasize visual modeling. In fact, it is quite the opposite. They call their approach to design "evolutionary design". It recommends incremental design, informally conceived (not using graphical diagrams, like UML ones), and directly expressed in the code. XP, for instance, does not forbid the elaboration of UML diagrams, but, in practice, discourages it [2].
XP states that the preferential way to communicate is talking, not graphic diagrams, although it accepts that documents are created in the end of the project, to register information useful to maintenance teams.
During a project, if the team is comfortable with diagrams, they can be informally used (on a blackboard, for instance), but not kept for future use nor updated.
Below, we briefly present some of the most important practices of XP.
Metaphor – it is considered by XP a synonym of software architecture. But while architecture is generally expressed graphically, a metaphor is expressed textually, like a story. It must define a coherent general theme that both customers and developers understand.
Refactoring – continuously redesign the software, to improve its response to change. Refactoring does not change software's functionality, but it's internal structure. When new functionality needs to be added, the first step is always refactoring, to simplify the functionality's increment.
Test first – develop some functionality tests before implement it. Automate these tests. After that, code the functionality and immediately apply the tests.
Continuous integration – build the software every two hours, so that it is possible to identify integration problems as soon as possible.
Martin Fowler [3] argues that the practices mentioned above – specially test first and continuous integration, but also refactoring – enable XP's evolutionary design approach, because they reduce the cost of changing the implemented code, neutralizing the perception that the cost of a change in the software grows exponentially along the software's life cycle. "With XP, it is possible to reduce the cost of changing, so that a change in any point of time will have the same cost" [8].

5 Analyzing the approaches presented

This section presents an analysis of strengths and weaknesses of each view presented before. The industry view will not be analyzed, because it only describes, in general, the actual situation and problems of software development.

5.1 The traditional Software Engineering Community view

Strengths

Following well-established software engineering best practices in a systematic way effectively increases the success rate (achieving cost and schedule targets, product quality) of software projects.
Jacobson's vision of a UML centered software development supported by powerful tools is gaining ground, although it still needs effort to become not just a vision, but a reality. OMG's vision goes in the same direction, prioritizing the use of UML and automatic transformation tools. Companies like IBM, Compuware and others are investing a lot to provide the adequate concepts and tools that will make that vision real.
Software projects of high size and complexity need a higher level of ceremony [5].
Prescriptive software engineering processes can be more flexible and dynamic than their detractors suppose. A process like RUP calls itself a process framework and, as such, needs to be configured for the specific needs and constraints of each project where it is used. There are light versions of RUP, of lower ceremony, more adequate for smaller and not so complex projects.

Weaknesses

For many big companies in which software is not the core business but represents a large part of the business (like financial institutions, for instance), developers frequently lack the culture and the expertise to build software models. [1] argues that UML modeling is impracticable for a whole generation of developers who will still be working for a long time.
Designing software, although conceptually and didactically very interesting, requires a large effort that is not feasible in many situations
Complex projects need specialized roles. Those who play the role of designers obviously focus on design. After some time, they loose contact with the most recent implementation techniques and resources, making their models – especially models with a lot of technological information – not respected by specialists in a specific technological platform. These will have their own ideas, frequently effectively more

evolved, about how to solve implementation problems [3].

5.2 The agile community view

Strengths

Questioning conventional wisdom is, in itself, strength, because it dismantles established dogmas and analyses the problem from a different point of view. The emphasis that the agile community put on that subject provides valuable insights for managers, developers and theorists.

Techniques like refactoring, test first, and others, have already proved their value.

Clearly defining the reason for the elaboration of each model (modeling for communication, modeling for documentation) underlines the cost/benefit of modeling and establishes clear rules to define modeling scope and its place in the development cycle.

Agile processes know that they are not useful in any situation. They clearly state that they should be used in not very big projects in which change is an important requirement.

Weaknesses

Delaying part of the modeling effort to the end of the project (modeling to communicate to the maintenance team) doesn't sound very good, because, in the end of the project, usually the team is committed with other projects or with solving problems of the present project. Not documenting at all is what usually happens if we think that modeling is just useful for documentation.

[3] criticizes XP's approach about the definition of a software architecture (that concept is absent of XP and is criticized by it). The establishment of software architecture in the initial phases of a project is useful, to stimulate the use of mature patterns and to define central and complex aspects that it would be difficult to change later. Using UML diagrams is useful for those comfortable with them.

[4] argues that the agile processes approach works well for highly qualified teams. Although the agile processes don't intend to reach all the developers community, this question is a weakness when we consider a broader perspective of the software industry, which can't count only on the professionals who are on the top of the pyramid.

The agile approach demands that developers have a wide and deep knowledge of the software code. Again, on a broader perspective, it is not feasible for big organizations, where there is always a turnover.

Some of the agile processes, like XP, are not very specific about how design is done. Others, like Agile Modeling [1] were created do fill that gap.

5 Conclusion

The traditional approach of the software engineering community, prioritizing visual modeling, effectively adds value to software development. Its best practices are conceptually incontestable. The benefits of visual modeling are increased if we explicitly define why the model is being built, like some in the agile community recommend.

Formalism not necessarily is weakness. It can be strength, for big and complex systems, and big and complex organizations.

On the other hand, the resistance of the software industry to adopt visual modeling in large scale means that, to a certain extent, the traditional vision failed. Software engineering needs to evolve, so that the gap between best practices and real practices can be reduced, and the success rate of software projects in industrial scale can increase.

Agile processes are valid in specific situations, but they also carry conception deficiencies, and are focused on a restrict universes: small to medium projects with development teams with very specific profiles.

In a short sentence: visual modeling and more prescriptive processes are useful do deal with complexity; while agile processes deal with the need of flexibility.

The MDD/MDA approach may still redefine the course of software engineering, like third generation languages did, some decades ago. More probably, though, it will be used in a more restrict universe (like agile processes). It is possible, too, that it fails, being adopted by very few organizations and projects. It still needs to prove its value.

References:

[1] Ambler, Scott. Agile Model Driven Development is Good Enough. IEEE Software. September-October 2003.

[2] Beck, Kent. Extreme Programming Explained. Addison-Wesley. 2000.

[3] Fowler, Martin. Is Design Dead? http://martinfowler.com/articles/designDead.html.

[4] Jacobson, Ivar. Not Every Light Process Is Agile. www.therationaledge.com.

[5] Kroll, Per; Kruchten, Philippe. The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP. Addison-Wesley. 2003.

[6] Greenfield, Jack; Short, Keith. Software Factories: Assembling Applications with Patterns,

Models, Frameworks, and Tools. Wiley Publishing Inc. 2004.
[7] Smith, John. A Comparison of RUP and XP. Rational Software White Paper. 2002.
[8] Astels, David; Miller, Granville; Novak, Miroslav. Extreme Programming: Guia Prático. Editora Campus, 2002.
[9] Lindvall, Mikael; Muthig, Dirk; Dagnino, Aldo; Wallin, Cristina; Stupperich, Michael; Kiefer, David; May, John; KähKönen, Tuomo. Agile Software Development in Large Organizations. IEEE Computer. December 2004. pg 26-33.
[10] [AM, 2005a]. Agile Modeling. Agile Modeling and eXtreme Programming. http://www.agilemodeling.com/essays/agileModelingXP.htm.
[11] [AM, 2005b]. Agile Modeling. Overview of The Values, Principles, and Practices of Agile Modeling (AM). http://www.agilemodeling.com/#ValuesPrinciplesPractices.
[12] MDA Guide Version 1.0.1. Object Management Group. Needham, Mass., EUA, july 2003. Available in <http://www.omg.org/cgi-bin/doc/omg/03-06-01.pdf>.