

# PARFAIT: Towards a Framework-based Agile Reengineering Process

Maria Istela Cagnin\*, José Carlos Maldonado,  
Fernão Stella R. Germano  
Universidade de São Paulo – USP  
Instituto de Ciências Matemáticas e de Computação  
São Carlos-SP, Brazil, CEP 13560-970  
{istela, jcmaldon, fernao}@icmc.usp.br

Rosângela Delloso Penteadó

Universidade Federal de São Carlos – UFSCar  
Departamento de Computação  
São Carlos-SP, Brazil, CEP 13.565-905  
rosangel@dc.ufscar.br

## Abstract

*The paper presents a sketch of a framework-based agile reengineering process, named PARFAIT<sup>1</sup>, whose objective is to provide the users with evolved versions of legacy systems, as soon as possible. The overall static structure of the Rational Unified Process (RUP), originally developed for forward systems engineering, has been here adapted for reengineering and is used for PARFAIT documentation. Frameworks are used in the process aiming at an agile approach to support the reengineering. Frameworks allow applications to be rapidly created, more than if they are built from scratch. Agile characteristics, such as incremental approach, cooperative approach with users and customers, straightforwardness, etc. give PARFAIT the ability to support the rapid evolution of the legacy system to a new version, according to the users and customers needs. A summary of a case study and the results obtained in the reengineering are presented. This study refers to a concrete reengineering case of a real system for controlling entry and exit of electronic appliances in a repair shop.*

**Keywords:** Agile Process, Reverse Engineering, Forward Engineering, Reengineering, Framework, Pattern Language.

## 1. Introduction

In 80's decade, with the advent of object oriented programming and visual languages, that facilitate the development of graphical user interfaces, several

organizations decided to migrate their legacy systems to such technologies. This migration is each day more intensive due to the maintenance difficulties of the legacy systems and the low usability of their textual interfaces.

Another factor that contributes to migration or evolution of legacy systems refers to the data storage form. Several organizations use file systems for data storage with no type of security, integrity, consistency or concurrency control. This causes high financial, operational and strategic risks, as the whole organization history is stored in a non-trustable form. A way to avoid these risks is the change of storage form to a DataBase Management System (DBMS). In addition, there has been a frequent change of platform. Many organizations are migrating their systems to open source platforms.

To ease the software maintenance and evolution activities to a new programming language/storage form/platform the Reengineering technique has been used. A relevant problem in using this technique is the lack of computing support, as the legacy systems usually have a large number of source code lines. Another important problem is the lack of a reengineering approach capable of evolving legacy systems in a rapid and efficient way, as these systems are vital to the organization and their quality assurance and correct functioning are indispensable to the main organization activities and to its permanence in a competitive market.

Nowadays, several processes, in the context of software development, called agile processes, are being proposed to allow the rapid development of systems in "Internet time": eXtreme Programming, Crystal Methods, Feature-Driven Development and Adaptive Software Development, all briefly described and compared relatively to their similarities and differences, by Abrahamsson et al [1].

From this panorama and from the framework capabilities to support systems development, we have devised the possibility of their utilization to support an agile reengineering process. This is due to the fact that

---

\* Financial support from FAPESP grant 00/10881-4

<sup>1</sup> PARFAIT is the acronym for *Processo Ágil de Reengenharia baseado em Framework no domínio de sistemas de Informação com VV&T* (in Portuguese), which means "Framework-based Agile Reengineering Process in the Information System Domain with VV&T".

frameworks offer time and effort economy because, according with Braga [5], they allow the reuse of large structures in a particular domain. It can be inferred that framework usage to support systems reengineering can ease these systems migration. Besides that, the product obtained after the reengineering process using a framework is prone to high quality, supposing that the framework had been validated during its construction.

This work intends to support the agile reengineering of legacy systems in the Business Resource Management domain, specifically of those that deal with resource location, trading or maintenance transactions, as for example, rental sores, libraries, hotels, etc. Some frameworks in that domain have been identified: IBM SanFrancisco [12], Gebos System [3], OminiBuilder [13] and GREN [5]. The last one is used to illustrate the main ideas explored in this paper.

In Section 2, the related works are discussed. In Section 3, the framework-based agile reengineering process sketch is presented. Several case studies have been conducted since the first definition of the reengineering process, and the results shall contribute to its refinement. In Section 4, a summary of a case study that has used the more recent version of the reengineering process to reengineer a real business legacy system to control an electronic repair shop is presented. In Section 5, the conclusions obtained and the future works to improve the reengineering process are presented.

## 2. Related Works

The object oriented reengineering of procedural legacy systems is considered by several authors. Gall & Klösh [21] propose a transformation process from procedural to OO code, COREM, which uses knowledge of the application domain. Penteadó [22] proposes a reverse engineering method – Fusion/RE to obtain OO analysis models from procedural legacy system. Costa [6] presents Fusion/RE-I, which is a method inspired on Fusion/RE concepts and ideas and supplies mechanisms to abstract functional and structural views from operational aspects and data available via the user interface. Battaglia et al. [23] propose the Renaissance method structured in two main phases. The first phase - *what to do* - aims to evaluate the organization and the legacy system, as well as to identify the need and urgency to conduct the reengineering and also the best strategy to carry it out. The second phase - *how to do it* - supports the planned transformation and guides all the migration process from the legacy system to the target system in an incremental form. Cimitile et al. [24] present a method to decompose legacy systems in objects. The object identification is centered in the storage of persistent data through files or relational database tables while programs and routines are candidates to implement methods.

There are some proposals of tools to support reengineering [11, 14, 16], but none of them explores framework support. As mentioned in Section 1, frameworks can be used to support reengineering, specifically in the forward engineering phase. From the case studies conducted, using PARFAIT, it was observed that frameworks inspire guidelines to the reverse engineering phase (because they guide the abstraction and retrieval of legacy system information).

A framework is a set of pre-fabricated software blocks that programmers can use, extend or adjust to build specific computing solutions. With frameworks, programmers do not have to start from scratch every time they write an application [15]. The GREN framework construction is based on the Business Resource Management pattern language – GRN<sup>2</sup> [4], which is presented in Section 2.1. According to Appleton [2], if a pattern is a problem solution in a certain context then a pattern language is a collection of such solutions that act together to solve a problem, according to a predefined objective. A pattern language can also be considered as a structured collection of patterns that transforms requirements and restrictions in software architecture [9].

Abrahamsson et al. [1] define an agile process as being: a) incremental (small software releases, with rapid cycles); b) cooperative (customer and developers working constantly together with close communication); c) straightforward (the method itself is well documented, easy to learn and to modify), and d) adaptive (able to make last moment changes). According to Turk et al. [17], the steps of agile processes and the project objectives are dynamically determined based on analysis of: (1) experience gained with previously executed process steps; (2) similar experience gained outside the project; and (3) changes in the requirements and development environment. The agility of a process is determined by the degree to which a project team can dynamically adapt the process based on changes in the environment and the collective experiences of the developers.

### 2.1 GRN Pattern Language

The GRN pattern language [4] is formed by fifteen patterns, some of which are applications or extensions of current patterns. GRN provides inexperienced developers enough information for developing new systems in the business resource management domain, together with alternative solutions.

GRN has a specific and well-defined domain, concentrated in the rental, trade and maintenance of business resources. For example, resource rental refers to

---

<sup>2</sup> GRN is the acronym for *Gerenciamento de Recursos de Negócios*, which means Business Resource Management in Portuguese.

the temporary usage of a good or service, as for example, a car rental or a specialist time usage. Resource trade refers to good property transference, as for example, a sale or an auction of products. Resource maintenance refers to the maintenance of a certain good, using manpower and parts for its execution, as for example, a house appliance repair in a specialized shop.

In Figure 1, the GRN pattern language patterns, their interdependencies and the order in which they are applied are shown. This language has three main patterns - RENT THE RESOURCE, TRADE THE RESOURCE, and MAINTAIN THE RESOURCE. The application of each one of these patterns and, consequently, of those that compose them, is done according to the objective of the application. The usage of these patterns is not mutually exclusive. For instance, in a mechanic vehicle shop that buys and sells parts, and also repairs cars. The patterns are divided into three groups: Group 1 (Business Resource Identification) has three patterns, that concern the business resources identification, quantification and (possible) storage; Group 2 (Business Transactions) has seven patterns, that are related to the management of the business resources identified in Group 1; and Group 3 (Business Transaction Detail) has five patterns, that concern the details involved in the transactions identified in Group 2. The GRN pattern language, expressed in UML (*Unified Modeling Language*) [10], is flexible because new attributes may be added according to the specific application.

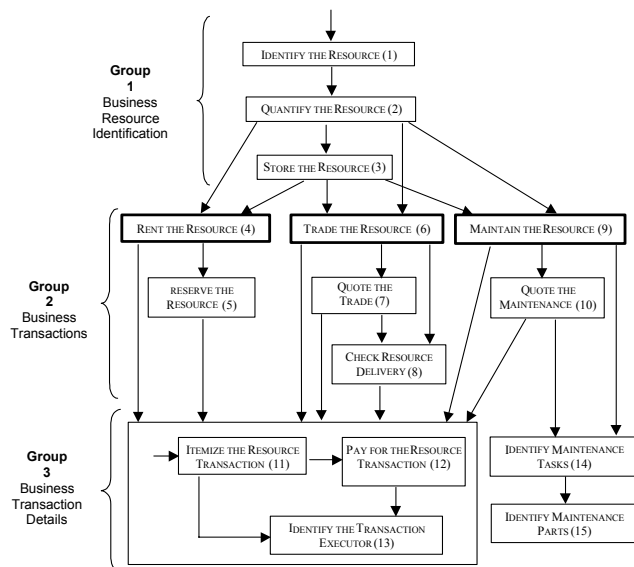


Figure 1 – Dependencies among patterns [4]

## 2.2 GREN Framework

The GREN framework [5] supports applications development in the business resource management domain. It was built based on the GRN pattern language

and is used in the reengineering process here described. This means that for each pattern of the pattern language, the corresponding classes in the framework are implemented. The programming language used in the GREN implementation is Smalltalk. Object storage is made in the MySQL [20] relational DBMS.

Other than the application classes, derived directly from the analysis models provided by the GRN pattern language, the GREN framework has also classes related to implementing data persistence and user interface. The GREN architecture is composed of three layers: persistence, application and user interface. The persistence layer handles the data storage in the database. The application layer handles the application functionality. The user interface layer handles the user interaction screens.

The creation of the application is conducted by the usage of the GRN pattern language, which results in a class diagram of the application instantiated. Based on these diagrams, GREN framework preprogrammed classes are used to create the new application code. GREN instantiation can be done in two ways: manual or helped by a wizard tool. In the first case, a document to help the framework instantiation (framework *cookbook*) is used. In the second case, the GREN-Wizard [5] tool provides facilities for selecting the patterns used and generates the new application classes and the MySQL database tables.

## 3. Framework-Based Agile Reengineering Process Sketch

This section presents the PARFAIT sketch that has the purpose of migrating small and medium procedural systems to the OO paradigm, in the business resource management domain. This process requires some basic knowledge for its application: i) of the GREN framework domain; ii) of the GRN pattern language; iii) of the Smalltalk programming language; iv) of the MySQL database and, v) of the application examples available, instantiated using the framework. According to Shull et al. [18], the understanding of application examples built by a framework is useful to support novice developers to produce systems more quickly through framework instantiation. This was confirmed by Braga [5] in an experiment that evaluated the GRN pattern language in the GREN framework instantiation. It is supposed that the deeper the knowledge of the software engineers in those items better the quality of the results of PARFAIT application.

The reengineering process documentation format is based on the RUP static structure, that describes **who** (workers), is doing **what** (artifacts), **how** (activities) and **when** (workflows) [19]. The Inception, Elaboration, Construction and Transition phases of the RUP process have been used in the reengineering process to group the activities with common purposes. Workflows specific to

PARFAIT have been created (Domain Understanding Framework/Legacy System, Requirements and Functionalities Understanding of the Legacy System, Analysis and Design Recovery, Business Modeling), some of RUP were adapted (Implementation, Test & Inspection) and others (Deployment, Software Configuration Management, Project Management and Planning, and Environment) were used without change, as presented in Table 1. The objectives of those workflows are distinguished according to the reverse engineering and forward engineering phases of the reengineering process.

The PARFAIT sketch is classified as incremental and interactive; furthermore counts with the users opinion and

approval during its application, so it considers a participative approach. This is very important because the users approve the product as the project evolves, and at each user interaction, product improvement is achieved. With this, the probability of the reengineered product satisfying the users needs and expectations is much higher than when they do not participate in the process. Anytime, during the process application, the software engineers can go back to any activity, in order to improve the product that is being generated. As the process uses an evolutionary approach, the documentation and the understanding of the legacy system are obtained in small chunks, according to the needs.

**Table 1 – Core Workflows of the reengineering process**

Core Workflows	Objectives
Domain Understanding Framework/Legacy System	<u>Reverse Engineering</u> : obtain information about the framework domain and the legacy system features <sup>3</sup> , to check the possibility of using the framework in the reengineering process. Framework non-functional aspects are also observed to determine their adequacy to the operational and financial aspects of the corporation that has requested the reengineering.
Requirements and Functionalities Understanding of the Legacy System	<u>Reverse Engineering</u> : establish and maintain contact with customers and other parties to determine what the legacy system does or should do and why; document system operations as use cases and describe the system requirements; document the input and output data used to execute the legacy system operations as test cases; supply test cases to discover legacy system business rules and functionalities not covered by the framework or offered by the framework but not required by the legacy system. <u>Forward Engineering</u> : give the process workers a better understanding of the legacy system requirements and functionalities; supply the test cases to test the forward engineering resulting system.
Analysis and Design Recovery	<u>Reverse Engineering</u> : translate the knowledge obtained from the legacy system functionality into a class diagram, with the help of the pattern language. The legacy system functionalities not covered by the pattern language have also to be modeled in the class diagram. <u>Forward Engineering</u> : supply the pattern language patterns used to support the framework instantiation.
Implementation	<u>Forward Engineering</u> : instantiate the framework with the patterns used in the class diagram development, to create an OO system prototype; adapt the prototype implementing the business rules and functionalities not covered by the framework; adapt the interface to that of the legacy system (if its necessary) and requested by the customer during the acceptance test; test each adaptation conducted with proper test cases.
Test & Inspection	<u>Reverse Engineering</u> : exercise the legacy system to understand its functionalities and document the test cases that will be applied to the OO system prototype; check whether all legacy system functionalities were properly implemented in the OO prototype; observe the test case coverage executed, in order to estimate the system quality; identify and guarantee that all defects discovered have been taken care of before system delivery; conduct inspections, for example using reading techniques, to support the validation of the artifacts produced in each activity. <u>Forward Engineering</u> : evaluate the prototype quality during its evolution through documented test cases, proceeding up to the final product evaluation.
Business Modeling	<u>Reverse Engineering</u> : identify the corporation business rules, implemented in the legacy system, through test cases; check with the users whether or not the business rules identified are true; document the business rules validated by the users. <u>Forward Engineering</u> : supply the business rules documented to facilitate the prototype adaptation.
Deployment	<u>Forward Engineering</u> : convert the legacy system database to that of the reengineered system; test the reengineered system in its final operational environment (beta test); install the reengineered system; train the final users.
Software Configuration Management	<u>Reverse Engineering and Forward Engineering</u> : track and maintain the integrity of the evolving project assets to allow the project team members to be able to localize artifacts, select the artifact proper version, observe the artifact history to understand its present state and the reasons for its alterations, and to know who is responsible for the artifact.
Project Management and Planning	<u>Reverse Engineering and Forward Engineering</u> : supply a structure to manage software-intensive large projects; supply practical guidelines for planning, staffing, executing and monitoring projects; supply a structure for risks management.
Environment	<u>Reverse Engineering and Forward Engineering</u> : select and allow tool acquisition; configure tools to process requirements; configure and improve the process, offer technical services to support the process (process application tools, backup, etc).

From Table 2 to Table 5 a summary of PARFAIT is presented, with the objectives of each process phase and the suggested time percentage for its execution; each phase activities and its objectives; each activity responsible worker, as well as an execution guide for the next activity/step. Metrics are being determined to observe the product quality and for process improvement. The activity steps are not presented in detail due to space restrictions.

The complete and detailed documentation of the PARFAIT sketch, together with a tutorial that shows its application step by step in a real case study, can be found in Cagnin et al. [7]. The guidelines to support each activity step, together with the inspections to validate the artifacts produced during the process application, can also be found in that document.

<sup>3</sup> Represent **what** the legacy system does in a higher abstraction level than the system operations.

**Table 2 – Reengineering inception phase summary**

<i>Phase: INCEPTION</i>			
<i>Objective:</i> Observe the risks to use the business resource management domain framework in a legacy system reengineering.			
<i>Suggested Percent Time:</i> 10			
<i>Essential Activities</i>	<i>Objective</i>	<i>Worker</i>	<i>Next Activity/Step</i>
Obtain familiarity with the framework domain	Analyze the domain and verify which features are covered by the framework.	Domain Analyst	<i>Activity:</i> Observe the legacy system domain in relation to the framework domain.
Observe the legacy system domain in relation to the framework domain	Identify the legacy system features to observe if it belongs to the same domain of the framework, in order to use the framework in the reengineering process.	Domain Analyst	<i>Activity:</i> Compare the non-functional framework features with those of the legacy system, or <i>Activity:</i> Develop a use case diagram to document the test cases.
Compare the non-functional framework features with those of the legacy system	Observe if the framework non-functional features are acceptable or modifiable to support the legacy system reengineering.	Domain Analyst	<i>Activity:</i> Develop a use case diagram to document the test cases.
<i>Milestone</i>	Evaluation criteria are established to check the viability of the framework usage in the reengineering process.		

**Table 3 – Reengineering elaboration phase summary**

<i>Phase: ELABORATION</i>			
<i>Objective:</i> Elaborate the legacy system documentation in order to support the <b>CONSTRUCTION</b> phase, in the framework instantiation and the adaptation of the prototype obtained.			
<i>Suggested Percent Time:</i> 30			
<i>Essential Activities</i>	<i>Objective</i>	<i>Worker</i>	<i>Next Activity/Step</i>
Develop a use case diagram to document the test cases	Understand the legacy system main functionalities and document the test cases used to exercise its operations.	System Analyst	<i>Activity:</i> Observe the legacy system domain in relation to the framework domain (if it is necessary to refine this activity), or <i>Activity:</i> Document the system business rules (if some business rule was identified during the legacy system use), or <i>Activity:</i> Develop the system class diagram
Develop the system class diagram	Develop a sketch of the legacy system class diagram based on the pattern language.	System Analyst	<i>Activity:</i> Document the changes done in the class diagram (after each update in the class diagram to represent the functionalities not covered by the pattern language), or <i>Activity:</i> Execute the test cases in the OO system prototype (in case that was the previously executed activity), or <i>Activity:</i> Develop an OO system prototype.
Document the changes done in the class diagram	Document the patterns used that do not fully fit the legacy system functionalities and also the addition of elements in the system class diagram.	System Analyst	<i>Activity:</i> Develop the system class diagram.
Document the system business rules	Document each business rule identified by the test cases.	System Analyst	<i>Activity:</i> Develop a use case diagram to document the test cases (in case that was the previously executed activity), or <i>Activity:</i> Execute the test cases in the OO system prototype (in case that was the previously executed activity).
Write the system user manual	Create the OO system user manual or a help <i>on-line</i> .	System Analyst	<i>Activity:</i> Convert the legacy system database.
<i>Milestone</i>	Evaluation criteria are established to verify if it is possible to obtain a proper general view of the legacy system; if the framework usage is really viable.		

**Table 4 – Reengineering construction phase summary**

<i>Phase: CONSTRUCTION</i>			
<i>Objective:</i> Create an OO system prototype and adapt it to make it functionally compatible with the legacy system.			
<i>Suggested Percent Time:</i> 45			
<i>Essential Activities</i>	<i>Objective</i>	<i>Worker</i>	<i>Next Activity/Step</i>
Develop an OO system prototype	Create an OO system prototype through the framework instantiation.	Developer	<i>Activity:</i> Execute the test cases in the OO system prototype.
Execute the test cases in the OO system prototype	Test the prototype with the test cases documented, in order to observe the difference of behavior among the system versions, so as to discover business rules and functionalities specific to the legacy system.	Tester	<i>Step:</i> Add to the class diagram the functionalities not covered by the pattern language patterns (if the functionality identified by the test case had not been documented), or <i>Activity:</i> Document the system business rules (when a business rule is identified by the test case), or <i>Activity:</i> Adapt the OO system prototype.
Adapt the OO system prototype	Adapt the prototype to the business rules, functionalities not covered by the pattern language, and to the users suggestions.	Developer	<i>Step:</i> Add, to the class diagram, the functionalities not covered by the pattern language patterns (after each change in the prototype), or <i>Activity:</i> Test the OO system (in case that was the previously executed activity), or <i>Activity:</i> Train the final users (in case that was the previously executed activity), or <i>Activity:</i> Write the system user manual (in case the implementation, of the adaptations previously defined, has been completed).
<i>Milestone</i>	Evaluation criteria are established to verify if the prototype satisfies the users modification requested and if it can be submitted to the validation test.		

**Table 5 – Reengineering transition phase summary**

<i>Phase: TRANSITION</i>			
<i>Objective: Assure that the software is ready to become available to the users and deploy it to in the organization.</i>			
<i>Suggested Percent Time: 15</i>			
<i>Essential Activities</i>	<i>Objective</i>	<i>Worker</i>	<i>Next Activity/Step</i>
Convert the legacy system database	Migrate the legacy system database data to that of the OO system.	Database administrator	<i>Activity: Test the OO system.</i>
Test the OO system	Submit the OO system to the test cases in order to evaluate the product maturity.	Tester	<i>Activity: Adapt the OO system prototype (if the test case result is different from the expected), or Activity: Train the final users.</i>
Train the final users	Prepare the users to properly use the OO system.	System Analyst	<i>Activity: Adapt the OO system prototype (if the user requested some change).</i>
<i>Milestone</i>	Evaluation criteria are established to evaluate the satisfaction of the users and of the organization that requested the reengineering.		

In Figure 2 an overview of the PARFAIT sketch is presented, highlighting its phases, its activities with an indication of mandatory execution, and its steps. The number inside in each circle, in each activity/step, represents the sequence of activities/steps carried out in a case study summarily presented in Section 4.

#### 4. Case Study

This section presents a summary of a case study using PARFAIT. It refers to a legacy system developed in Clipper, with about 4.800 source code lines, used to manage the entry and exit of electronic appliances in a specialized shop. The system has 26 operations for supporting registration, processing, reports, and queries.

The activities *Obtain familiarity with the framework domain*, *Observe the legacy system domain in relation to the framework domain*, *Compare the non-functional framework features with those of the legacy system* and *Develop a use case diagram to document the test cases* have been performed sequentially as shown in the circles numbered 1 to 4 in Figure 2.

After use case diagram construction, users of the legacy system validated it, so as to verify if there were new requirements, inherent to the organization functioning, to be added or others to be removed. The users have approved this diagram and the reengineering project could proceed.

Considering the general overview of the PARFAIT sketch of Figure 2, the following activities *Develop the system class diagram* and *Document the changes done in the class diagram* have been executed iteratively as shown in the circles numbered 5 to 11 in Figure 2. This shows that there have been a gradual evolution and refinement of the class diagram in the last version presented in Figure 3. Classes *Technician*, *Customer*, *WorkOrder*, *Appliance*, *Type* and *TypeSpecification* result of classes instantiation of the patterns (1) IDENTIFY THE RESOURCE, (8) MAINTAIN THE RESOURCE, and (14) IDENTIFY THE TRANSACTION EXECUTOR of the GRN

pattern language and have attributes and methods specific of the legacy system. Class *System* has been created to implement the system generic functionalities, as for example, label emission with the company data.

After GREN framework instantiation (activity *Develop an OO system prototype*) supported by the GREN-Wizard instantiation tool [5] and conducted based on the GRN patterns used during the class diagram development (activity *Develop the system class diagram*), the OO system prototype could be exercised. Next, the test cases, exercised in the legacy system and documented in the activity *Develop a use case diagram to document the test cases*, have been re-applied to the OO system prototype (activity *Execute the test cases in the OO system prototype*) to discover business rules and functionalities specific of the legacy system or present in the framework and not required by the legacy system. As the system is small and belongs to a specific domain, only one business rule and seven functionalities have been identified through test cases. The business rule identified has been validated by the users and refers to the guarantee of the repair executed in the electronic shop (if the last repair exit data of a certain appliance occurred in the last 90 days, then the appliance repair is inside the guarantee period). Of the seven functionalities identified, five are present in the legacy system and are not present in the OO system prototype and two are present in the OO system prototype but are not required by the legacy system, because they have been inherited from the framework. It is important to mention that no new functionality has been added to the system. After the business rule discovery, its documentation has been made in the activity *Document the system business rules* and, subsequently, both the business rule and the functionalities have been implemented in the prototype through the activity *Adapt the OO system prototype*. Each prototype adaptation has been tested with the test cases that identified the reason for the adaptation (business rule or functionalities) and the developer, according to the needs, has created other test cases.

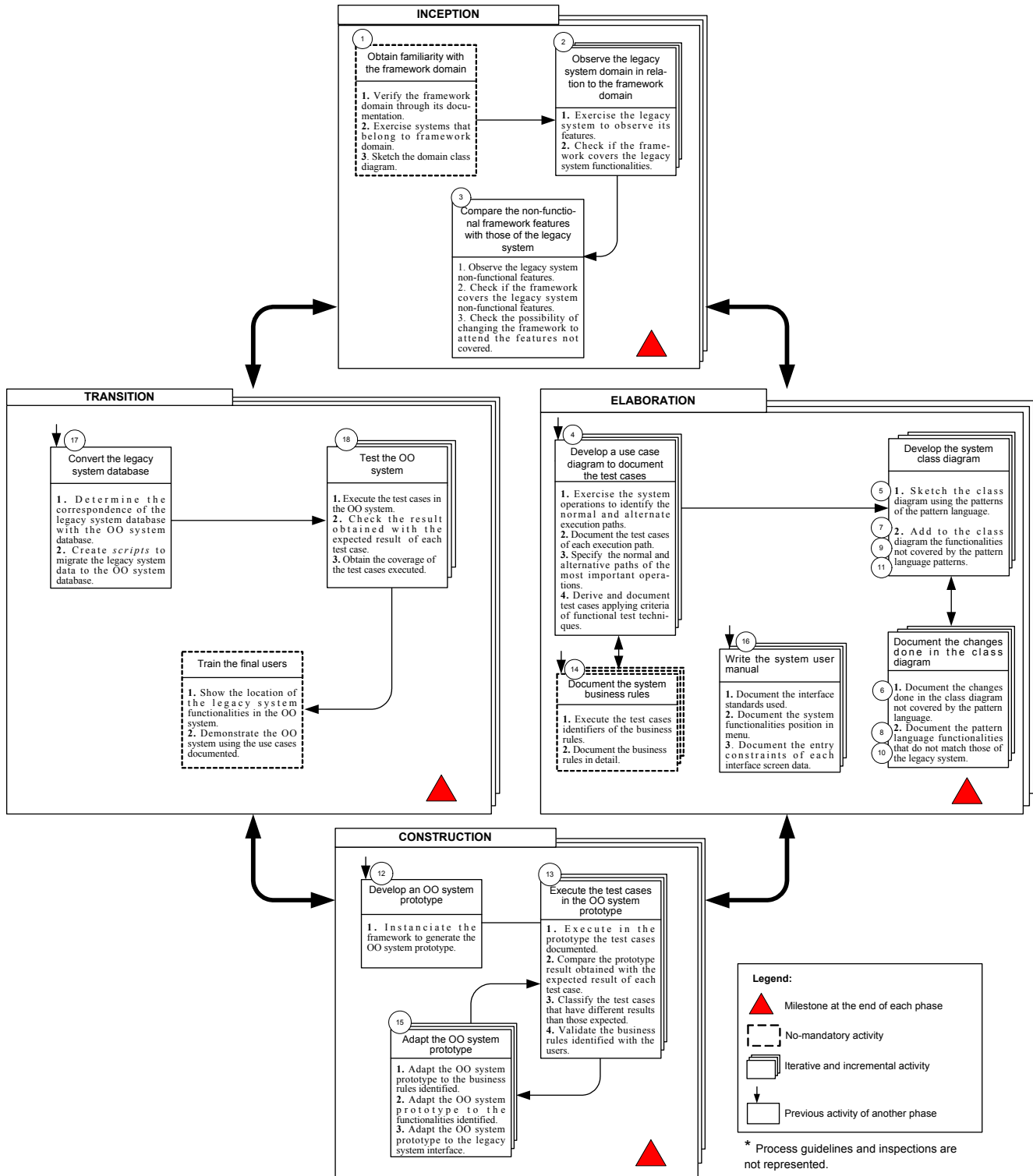


Figure 2 – Overview of PARFAIT

After conducting adaptations in the prototype, it has been presented to the user and exercised together with the legacy system. This allowed the user to clearly observe the difference between the two version interfaces and to suggest improvements (for example, modification in the result of two queries; modification in the form of the electronic appliance identification in the OO system; implement alternative to combo-box components, when there is the possibility of loading a large number of records, so as to avoid system degradation, etc). These suggestions allowed the identification of twelve new functionalities and four new framework *hotspots* (that is, the suggestions can be considered generic and useful to other applications). The suggestions have been classified according to their priority and the major part has been implemented. After completing the adaptations requested by the users, a system manual was written (activity *Write the system user manual*).

The legacy system database has been converted (activity *Convert the legacy system database*) to the MySQL database management system, used by the OO system. Then, the activity *Test the OO system* has been executed using the test cases derived from the legacy system, in order to observe the functional similarity between the system versions. After system deployment, the faults detected during the initial system utilization

period can be recorded in order to analyze the product quality and to subsidize eventual process improvement activities.

In Table 6, the differences between the legacy system and the OO system, resulting of the reengineering relative to some features, are presented. The large difference in LOC between the legacy system and the OO system is due to the fact that, in the last, there is reuse of the framework methods that implement the domain inherent functionalities.

According with Table 6, out of the 16 Smalltalk system classes, only one has been manually created and the others have been instantiated through the framework; out of the 290 object oriented system methods, 218 methods have been obtained by the framework instantiation. The other methods have been manually implemented and of these, 61 methods have been created from scratch, 2 methods have resulted of the modification of existing methods, and 9 methods were overridden. The main reason why there are more methods in the OO system than in the legacy system is that the last has not data encapsulation and several procedures implement more than one functionality.

Figure 4 shows one of the legacy system screen and Figure 5 shows the corresponding screen in the OO system after applying PARFAIT.

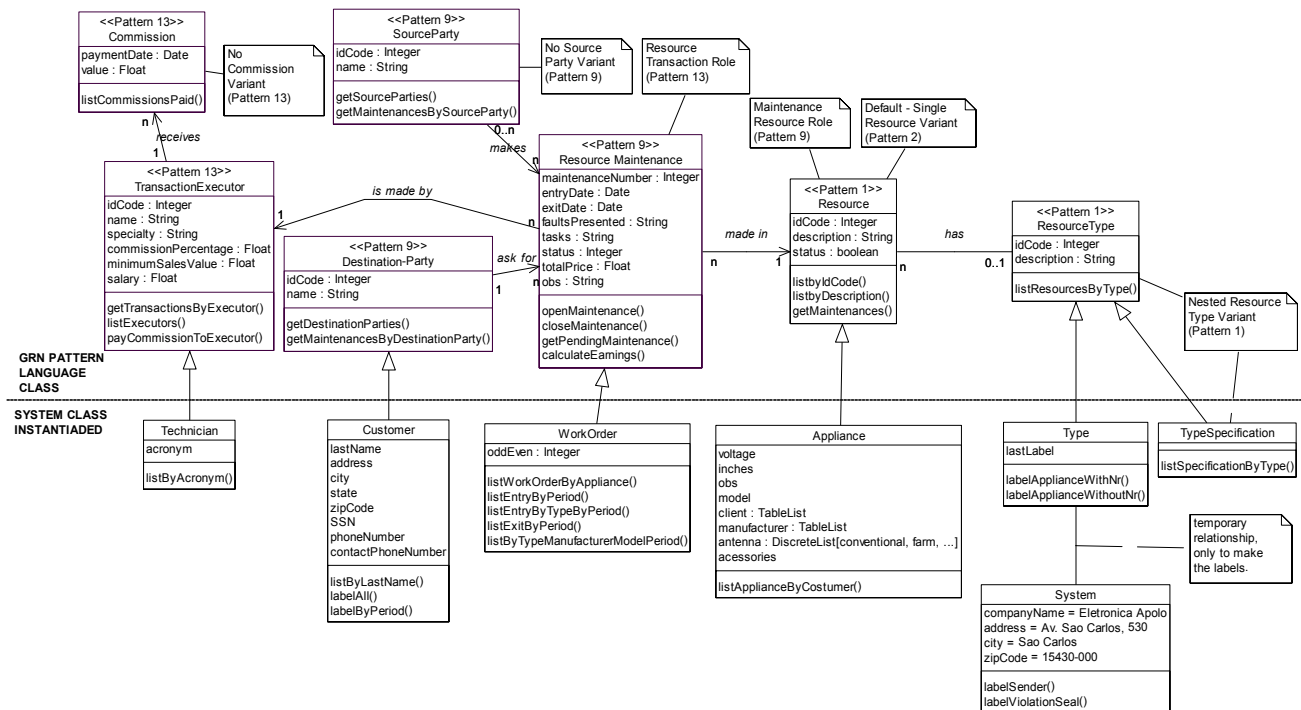
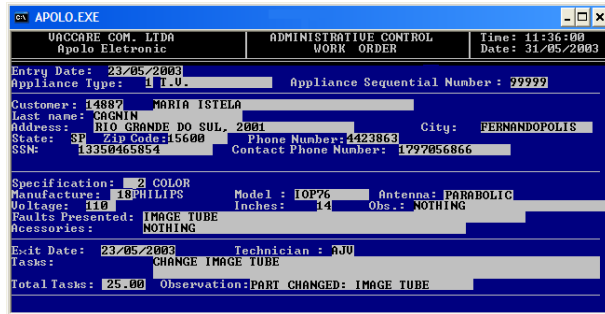


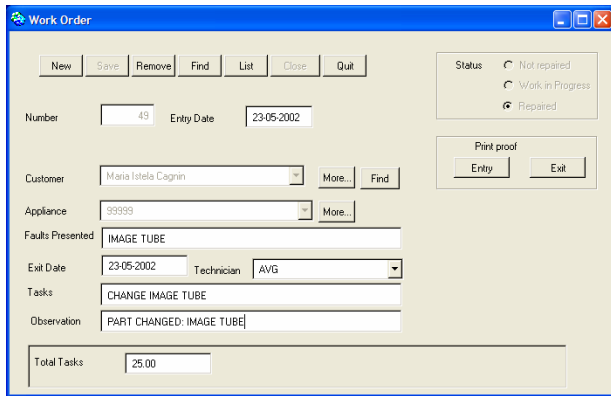
Figure 3 – Class Diagram – final version

**Table 6 – Features of the legacy system versus OO system**

Features	Legacy System	OO System
programming language	Clipper	Smalltalk
number of modules/classes	30	16
number of procedures /methods	63	290
number of business rules	1	1
number of test cases	16	#40
LOC	#4.800	#2.200
development time	#30 days <sup>4</sup>	#10 days



**Figure 4 – Legacy System Screen**



**Figure 5 - OO System Screen System**

## 5. Conclusion and Future Work

This paper presented a framework-based agile reengineering process sketch, named PARFAIT, which uses a pattern language in which the framework construction has been based, to recover the legacy system class diagram, in order to facilitate the framework instantiation, providing a prototype at the very initial steps of the reengineering process. Furthermore, it documents test cases resulting of the legacy system execution and use them: (a) to apply them to the evolving prototype resulting of the framework instantiation, to discover business rules

and functionalities specific of the legacy system or present in the framework and not required by the legacy system and (b) to test the OO system to verify the similarity of its functionalities with those of the legacy system.

The system resulting of the PARFAIT sketch can be considered just a prototype in case there are, for example, non-functional requirements (performance, consistency, security, reliability, etc) that have not been fully satisfied. In that case, it is necessary that such prototype be submitted to evolutionary processes or reengineering processes keeping the OO paradigm until it fully satisfies the initially requested requirements.

From the analysis of the PARFAIT sketch features and from its experimentation, it has been observed that the greatest majority of the agile processes features [1, 8, 17] are present in it: a) it has an incremental approach; b) it is cooperative, as it counts with users and customers opinions and approvals during the process application, and these actively participate of the decisions made during the project, cooperating for the product to be approved as the project evolves; c) it is straightforward, as the process is easy to understand and has a complete documentation in tutorial form; d) it is adaptive, as new user requirements can be added at any moment during its application; e) it allows the software engineer to obtain an executable version of the legacy system in the OO paradigm as soon as possible; f) it constantly tests the product; and g) the execution of its activities is dynamic, that is, can or cannot be executed, depending on the experience of the software engineers that are applying the process.

By the experience obtained from case studies conducted, it was observed that a significant amount of the time and effort have been spent in the prototype adaptation and the largest difficulty faced was in the understanding of the framework structure to make correct the adaptations in the prototype.

The domain specific framework does not impose constraint in the system size limit. The main points related to scalability of this approach are the effort for understanding the system and the effort to adapt and/or evolve the framework to attend the requirements. These points will be addressed in the next case study we are planning to carry out.

At this stage, addressing the non-functional requirements is not our main concern, although some are addressed indirectly such as usability for the framework somehow establishes a common interface amongst the applications. Performance, security and portability are not addressed at all, at this stage.

Experiments shall be planned and conducted in order to evaluate and improve the quality and maturity of PARFAIT. A tool to support the process application will be defined and implemented aiming at improving its agility and also to guarantee the management of the

<sup>4</sup> it was obtained with the legacy system developer

produced artifacts versions in the several iterations performed during PARFAIT application.

## References

- [1] Abrahamsson, P.; Salo, O.; Ronkainen, J.; Warsta, J. *Agile Software Development Methods. Review and Analysis*. ESPOO (Technical Research Centre of Finland) 2002. VTT Publications n. 478, 107 p. URL:<http://www.inf.vtt.fi/pdf/publications/2002/P478.pdf>. Accessed: November, 2002.
- [2] Appleton, B. *Patterns and Software: Essential Concepts and Terminology*, 1997 URL: <http://www.enteract.com/~bradapp/docs/patterns-intro.html>. Accessed: February, 2002.
- [3] Bäumer, D.; Gryczan, G.; Knoll, R.; Lilienthal, C.; Riehle, D.; Züllighoven, H. *Framework Development for Large Systems*. Communications of the ACM, v. 40, n. 10, p. 52-59, October, 1997.
- [4] Braga, R.T.V.; Germano, F. S. R.; Masiero, P. C. *A Pattern Language for Business Resource Management*. In: Annual Conference on Pattern Languages of Programs, PLOP'99, 6, Monticello, Illinois, EUA, Proceedings, v.7, p. 1-33, August, 1999.
- [5] Braga, 2003. *A Process for Construction and Instantiation of Frameworks based on a Domain-Specific Patterns Language*. Sc.D. Thesis – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos-SP, Brazil, 224 p. (in portuguese).
- [6] Costa, R. M. (1997). *Fusion-RE/I: A Reverse Engineering Method to Support Software Maintenance*. M.Sc. Dissertation. Instituto de Ciências Matemáticas e de Computação, ICMC-USP, São Carlos-SP, Brazil, 112 p. (in portuguese).
- [7] Cagnin, M.I.; Maldonado, J.C.; Penteado, R.; Germano, F. *Reengineering Process Based on Framework: Definition and Application Example*. Working Document. ICMC-USP. São Carlos-SP, Brazil, November, 2002.
- [8] Coad, P.; Palmer, S. *Feature-Driven Development: Guide*. URL: <http://www.nebulon.com/articles/fdd/latestfdd.html>. Accessed: November, 2002.
- [9] Coplien, J. O. *Software Design Patterns: Common Questions and Answers*, in Linda Rising (editor) *The Patterns Handbook: Techniques, Strategies, and Applications*, Cambridge University Press, New York, p. 311-320, 1998.
- [10] Fowler, M., Scott, K. *UML Distilled – Applying the Standard Object Modeling Language*. Publisher Addison-Wesley, First Edition, 1997.
- [11] Ghannouchi, S.A.; Ghezala, H. H. and Kamoun, F. *A Generic Approach for Data Reverse Engineering Taking into Account Application Domain Knowledge*. In: Euromicro Conference on Software Maintenance and Reengineering, CSMR'98, 2, Florence, Italy, Proceedings, p. 21-28, 1998.
- [12] IBM. Introduction to SanFrancisco. URL: <http://www-3.ibm.com/software/ad/sanfrancisco/concepts/ibmsf.sf.TSFConceptsAndFacilities.html>. Accessed: February, 2002.
- [13] Omnisphere Information Systems Corp. *OmniBuilder – Open Application Generator*. URL: <http://www.omni-sphere.com/overview/overview.htm>. Accessed: February, 2002.
- [14] Sneed, H. M.; Nyáry E. *Extracting Object-Oriented Specification from Procedurally Oriented Programs*. In: Working Conference on Reverse Engineering, WCRE'95, 2, Toronto-Canada. Proceedings. IEEE Computer Society Press, p. 217-226, 1995.
- [15] Taligent Inc. *Building Object-Oriented Frameworks*. URL:<http://www-106.ibm.com/developerworks/java/library/oobuilding/?dwzone=java>. Accessed: February, 2002.
- [16] Taschwer, M.; Rauner-Reithmayer, D.; Mittermeir, R. *Generating Objects from C Code – Features of the CORET Tool-Set*. In: European Conference on Software Maintenance and Reengineering, CSMR'99, 3, Amsterdam-The Netherlands, Proceedings, IEEE, p. 91-100, 1999.
- [17] Turk, D.; France, R.; Rumpe, B. *Limitations of Agile Software Processes*. In: 3<sup>rd</sup> International Conference on Extreme Programming and Agile Processes in Software Engineering (XP'2002), Alghero, Sardinia, Italy, p. 43-46, May, 2002.
- [18] Shull, F.; Lanubile, F.; Basili, V. (2000). *Investigating Reading Techniques for Object-Oriented Framework Learning*. IEEE Trans. on Software Engineering, v. 26, n. 11, November.
- [19] Kruchten, P. (2000). *The Rational Unified Process: An Introduction*. Second Edition, Addison-Wesley, 298 p.
- [20] MySQL (2002). *MySQL Reference Manual for Version 3.23*. URL: <http://web.mysql.com/>. Accessed: February, 2002.
- [21] Gall, H., Klösch R. (1993). *Capsule Oriented Reverse Engineering for Software Reuse*. In: European Software Engineering Conference, ESEC'93, 4, Garmish-Partenkirchen, Germany, Proceedings, p. 418-33.
- [22] Penteado, R. A. D. (1996). *A Method for Object Oriented Reverse Engineering*. Sc.D. Thesis – Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos-SP, Brazil, 237 p. (in portuguese).
- [23] Battaglia M.; Savoia, G.; Favaro, J. (1998). *Renaissance: A Method to Migrate from Legacy to Immortal Software Systems*. In: Euromicro Conference on Software Maintenance and Reengineering, CSMR'98, 2, Florence, Italy, Proceedings, p. 197-200.
- [24] Cimitile, A.; De Lucia, A.; Di Lucca, G. A.; Fasolino, A. R. (1999). *Identifying objects in legacy systems using design metrics*. The Journal of Systems and Software, n. 44, p. 199-211.