An Agile Reverse Engineering Process based on a Framework

Maria Istela Cagnin *[1], José Carlos Maldonado[1], Fernão Stella R. Germano[1], Paulo Cesar Masiero[1], Alessandra Chan ♦[1], Rosângela Dellosso Penteado[2]

[1]Universidade de São Paulo – USP
Instituto de Ciências Matemáticas e de Computação
São Carlos-SP, Brazil, CEP 13560-970
{istela, jcmaldon, fernao, masiero}@icmc.usp.br
ale@grad.icmc.usp.br
[2]Universidade Federal de São Carlos – UFSCar
Departamento de Computação
São Carlos-SP, Brazil, CEP 13.565-905
rosangel@dc.ufscar.br

Abstract. *This paper presents an agile reverse engineering process, referred to as PARFAIT/RE, which has been abstracted from the use of a framework-based agile reengineering process, named PARFAIT[3]. The proposition of PARFAIT/RE has been evidenced from an analysis done in a reengineering case study of a medium size system. Several factors collaborate to make PARFAIT/RE agile: a) active participation of legacy system users to validate the artifacts created and to suggest new requirements or the removal of requirements that do not belong to the business context; b) incremental approach; c) prototyping paradigm feasible from the framework instantiation; and d) use of an analysis pattern language (GRN), which was the basis for building the GREN framework. GRN belongs to the business resource management domain and supports object oriented modeling of procedural legacy systems. The analysis models created are used to support the framework instantiation.*

1 Introduction

The concern of organizations with regards to evolving their legacy systems, so as to assure their competitiveness in the current market, is remarkable. On the other side, there is resistance in the transition to a new system, because any problem or non-

* Financial support from FAPESP grant 00/10881-4
♦ Financial support from CNPq

[3] 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”.

foreseen occurrence may put the business at risk, and may even lead it into bankruptcy. For legacy systems evolution, there are three alternatives: 1) development of a new system, 2) reengineering of the legacy system, or 3) legacy system documentation recovery to support maintenance activities. This paper aims to supply support to the third alternative in a rapid and efficient way.

The traditional reverse engineering approaches do not provide an agile modeling [5] because: a) several types of artifacts are supplied, some may be unnecessary, and others may contain irrelevant information; b) possible lack of effective user participation during the whole reverse engineering both to validate the models produced and to help software engineers to better understand the requirements; c) possible lack of an effective approach that supports the domain knowledge, the recovery of existing requirements or the elicitation of new ones; etc.

From the analysis of this situation, the possibility of using PARFAIT – an agile reengineering process [6, 7] - to abstract an agile reverse engineering process has been noticed. Considering the features of agile methodologies and techniques [1, 18, 5] of PARFAIT, the reverse engineering process based on a prospective case study, referred to as PARFAIT/RE, was abstracted. PARFAIT/RE is considered agile, because it is possible to deliver a light, updated and validated documentation of the legacy system, very early in the reverse engineering process, necessary for maintenance people to evolve systems more effectively. It is important to mention that the PARFAIT process has been initially created to support system reengineering, without distinguishing reverse and forward engineering phases.

The aim of this paper is to show a summary of PARFAIT/RE activities, insights and lessons learned with the reengineering prospective case study carried out.

In Section 2, the related work is discussed. In Section 3 PARFAIT is presented. In Section 4, the abstraction of the PARFAIT/RE agile reverse engineering process based on a reengineering prospective case study is described and at the end of that section a PARFAIT/RE activities summary is presented. In Section 5 the conclusions and suggestion for future works are discussed.

2 Related Work

The object oriented reverse engineering of procedural legacy systems is considered by several authors. Penteado proposes a reverse engineering method – Fusion/RE to obtain OO analysis models from the analysis of procedural legacy code [11]. Costa 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 [13]. Cimitile et al. present a method to decompose legacy systems in objects [14]. The object identification is centered on the storage of persistent data through files or relational database tables, while programs and routines are candidates to implement methods. None of these approaches bear on user participation during reverse engineering. None supply knowledge on the domain in which the legacy system is inserted and none supply an effective way to elicit new user's requirements.

According to Pressman [10], the prototyping paradigm is used for requirements gathering and can be very effective for that purpose. That paradigm is used by PARFAIT/RE to support requirements elicitation.

Several tools can be used to support the prototyping paradigm. One of them is a framework because it allows applications to be rapidly created, more than if they are built from scratch. A framework is a set of pre-fabricated software blocks that programmers can use, extend or adjust to build specific computing solutions [15]. The framework GREN [3] is being used in PARFAIT/RE for that purpose.

GREN supports applications development in the business resource management domain. It was built based on the GRN pattern language [4]. The programming language used to implement GREN is Smalltalk [2] and the object storage is made in the MySQL [17] relational DBMS. The creation of the prototype is conducted using the GRN pattern language, which results in a class diagram of the legacy system abstraction. Based on these diagrams, GREN framework preprogrammed classes are used to create the prototype code. GREN instantiation can be done through GREN-Wizard [3], a tool that provides facilities for selecting the patterns used and generates the new prototype classes and the MySQL database tables.

The GRN pattern language is formed by fifteen analysis patterns that provide inexperienced developers enough information for developing new systems in the business resource management domain, together with alternative solutions. This language has a specific and well-defined domain, concentrated in the rental, trade and maintenance of business resources and is expressed in UML (*Unified Modeling Language*) [21].

In many software engineering situations, the software engineer's knowledge of the domain is incomplete and he/she may have to learn more about the domain from the system code [20]. But this task is hard to do when the system is obsolete and has passed through many maintenance activities. An analysis pattern language, that belongs to the same domain as the legacy system, supplies the software engineer with the necessary knowledge about the domain, without demanding him to visit the code, saving time and effort. That advantage and the support to prepare the OO documentation of the legacy system have been observed during some uses of PARFAIT [7, 8].

A way to analyze an agile reverse engineering approach is observing if it fits the Agile Modeling (AM) principles. AM is a practice-based process that describes how to be an effective modeler. Current modeling approaches can often prove dysfunctional. In the one extreme, modeling is non-existent, often resulting in significant rework when the software proves to be poorly thought through. The other extreme is when excessive models and documents are produced, which slows the development efforts down to a snail's pace [5].

3 PARFAIT Agile Reengineering Process

The main purpose of the PARFAIT agile reengineering process [6, 7] is to migrate small and medium size procedural systems to the object oriented approach and assure that the software product, resulting from the reengineering process, is reliable and accepted by the users. With this purpose, it uses i) the GREN framework as computer-aid, which builds the system prototype, facilitating the requirements elicitation and the migration of the legacy systems to the Smalltalk language and MySQL DBMS; ii) GRN pattern language, that supports legacy system documentation, elaboration, and knowledge about the legacy system domain; iii) users interaction during the process application so that the product is evaluated as it evolves, and; iv) functional tests [16], applied during the process.

The framework based computer-aid is an important issue to contribute to the process agility, as it provides a system prototype as soon as possible. This prototype evolves during the PARFAIT process application through successive versions, until it reaches the definitive system.

PARFAIT supports testing activities to find the business rules and specific functionalities of the system, as well as to validate the system produced. For this, it uses, in its current version, the functional criteria Equivalence Classes Partitioning and Boundary Value Analysis [16]. The process documentation is based on the static structure provided by RUP [12].

In Fig. 1**,** the PARFAIT process is illustrated with the activities that are performed during the Inception, Elaboration, Construction, and Transition phases. The original objectives of these phases have been replaced by reengineering specific objectives.

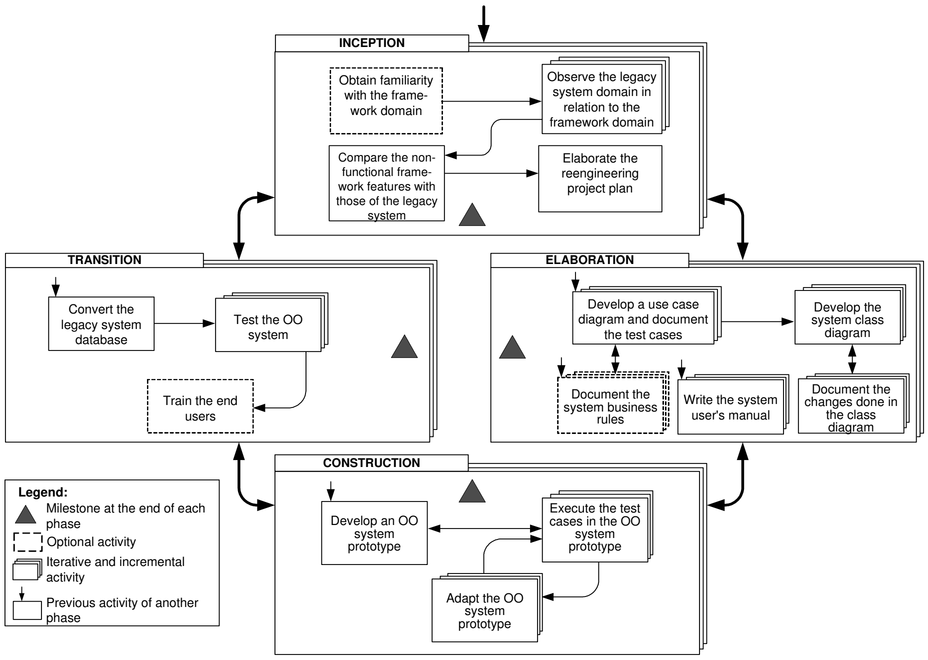

INCEPTION
Obtain familiarity with the frame-work domain
Observe the legacy system domain in relation to the framework domain
Compare the non-functional frame-work features with those of the legacy system
Elaborate the reengineering project plan
TRANSITION
Convert the legacy system database
Test the OO system
Train the end users
ELABORATION
Develop a use case diagram and document the test cases
Develop the system class diagram
Document the system business rules
Write the system user's manual
Document the changes done in the class diagram
CONSTRUCTION
Develop an OO system prototype
Execute the test cases in the OO system prototype
Adapt the OO system prototype
Legend:
Milestone at the end of each phase
Optional activity
Iterative and incremental activity
Previous activity of another phase

Fig. 1. PARFAIT overview

It is important to stress that the software engineer may return to any of the process activities, at any phase, in order to refine artifacts already produced.

In PARFAIT, there are some optional activities, and their execution should be decided by the person responsible for the reengineering project. The activities may pass through several iterations.

Before moving from a phase to another, during process activities iterations, it is necessary to perform verifications by means of milestones, represented by triangles on Fig. 1. Their objective is to evaluate the reengineering process progress, deciding to continue or stop its application.

After completing each artifact, verifications and validations are carried out. These may be performed applying "ad hoc" techniques or using the OORT's [19] reading technique. The last case must be applied only to some artifacts produced using UML notation. The artifacts with easy representation are also validated by users (for example: requirements document, use case diagrams, etc).

After the conclusion of each artifact produced by an activity, the software configuration management is performed, so that its versions may be recovered at any time, during and/or after the conclusion of the reengineering project. A version control management tool must be used to support this task.

4 PARFAIT/RE: an Agile Reverse Engineering Process

The PARFAIT/RE process, summarized at the end of this section, has been abstracted from a reengineering prospective case study of a medium size legacy system using the PARFAIT process. It has been observed that iterations in certain activities of PARFAIT are not necessary when one wants to do only reverse engineering.

The system submitted to reengineering is an University library control system, implemented in Clipper with about 6 KLOC. Further information about this reengineering case study is available in [8].

A user of the library control legacy system (member of the University staff) has participated of the whole process conduction. This allowed the quality evaluation, although superficial, of the reengineering process product results. Product evaluation with different user types (library attendants, students, teachers, etc), both of the legacy system and of other library systems, shall be conducted for the product quality to be more efficiently evaluated.

Information on the team, responsible for the PARFAIT application, has been omitted, because the process was applied individually by a software engineer.

Table 1 shows the history of PARFAIT activities iterations performed during the reengineering prospective case study, highlighting with shades those that should be done when the interest is to do only reverse engineering. These activities, as well as the others activities that compose PARFAIT, will be commented on during the presentation of the case study.

It has been observed that the two initial activities of PARFAIT Inception phase are conducted independently of the form in which the process is used, that is, either for reengineering or to conduct reverse engineering, as they supply the context of the legacy system domain in relation to that of the GREN framework.

After running the legacy system for a short time in the activity *"Observe the legacy system domain in relation to the framework domain"*, it was observed that the GREN framework is useful to support the reverse engineering of the library control legacy system. This is because the main transaction of the legacy system is the book rental control. In this activity, the elaboration of the artifact *"Requirements document"* has been started, establishing the system general objectives.

Table 1. History of reengineering case study activities iterations

Iterations numbers	***Activities***
1	Obtain familiarity with the framework domain
2	Observe the legacy system domain in relation to the framework domain
3	Compare the non-functional framework features with those of the legacy system
4	Elaborate the reengineering project plan
5, 18, 25	Develop an use case diagram and document the test cases
6, 8, 10,15,22	Develop the system class diagram
7, 9,11,16,23	Document the changes done in the class diagram
12	Develop an OO system prototype
13, 19, 26	Execute the test cases in the OO system prototype
14, 21	Document the system business rules
17, 20, 24	Adapt the OO system prototype
27	Write the system user's manual
28	Convert the legacy system database
29	Test the OO system
not performed	Train the end users

The two following activities "*Compare the non-functional framework features with those of the legacy system"* and *"Elaborate the reengineering project plan"* should not be done, as they aim to satisfy specific features of the system reengineering. Due to space limitation, only the last version of the artifacts produced is shown in this paper.

In the Elaboration phase, several iterations were conducted to complete the activity *"Develop a use case diagram and document the test cases"*. In this activity, it was observed that a long time was spent (552:50 hours), mainly on the artifacts related to test: 6 hours to produce the artifact *"Use case diagram"*, 500 hours to produce the artifact *"Test case documentation"*, 36 hours to produce the artifact *"Equivalence class documentation"* (Table 2), 5 hours to produce the artifact *"Data dependence diagram among the use cases"* and 5:50 hours to complete the artifact *"Requirements document"*, started on the Inception phase. In the artifact *"Equivalence class documentation",* 174 classes were created, enumerated as shown in Table 2, and in the artifact *"Test case documentation",* 354 test cases were documented. The majority of these test cases were generated from the equivalence classes documented.

The artifacts *"Use case diagram"* and *"Requirements document"* were created as the legacy system was executed. In this case study, the worker[4] has described initially

[4] Person that executes the activity, according with RUP notation.

all the system features as functional requirements and then, grouped them by concerns[5], as can be seen on Table 3. The non-functional requirements have been obtained through interviews with the user and were documented in the artifact *"Requirements document"* in the same form of the functional requirements. The user effectively participated in the validation of the artifacts *"Use case diagram"* and *"Requirements document"*.

Table 2. Fragment of the equivalence class documentation for the book loan feature

Input Restrictions	***Valid Class***	***Invalid Class***
Student code	Sequence of 8 numeric characters **(1)**	Sequence of characters **(2)** Different sequence of 8 numeric characters **(3)**
Existence of student code	Existence **(4)**	Non-existence **(5)**
Exemplar code	Sequence of up to 6 numeric characters **(8)**	Sequence of character **(9)** Sequence of more than 6 numeric characters **(162)**
Status of the exemplar	Book not loaned **(10)**	Book loaned **(11)**
Loan date	Day and month valid **(16)** 0100 < year <= 2999 **(18)**	Day and month not valid **(17)** Year < 0100[6] **(19)**
…	…	…

Table 3. Fragment of the requirements document for the library system

REQUIREMENTS DOCUMENT
<u>**General Objectives**</u>**:** The library control system aims to manage the circulations of the collection. <u>**Functional Requirements**</u>**:** **Concern: Book** 1. The system should record the insertion, alteration and removal of new books. Data related to the title, subtitle, U.D.C. (Universal Decimal Classification) class, P.H.A. (Author Classification Table) class and area will be recorded. The authors and subjects contained in the book and its code representation on the system are also listed. 2. The system should record the insertion, change and removal of the new book exemplar related to a book previously recorded that now will be resources to be lent. 3. … 12. The system should provide the title of the books recorded in the system, besides showing the book recording data, i.e. title, subtitle, U.D.C. class, P.H.A. class, area, authors, sub-area and subjects. **Concern: Student** 13. The system should record the insertion, change and removal of the new students who will be the customers who will borrow book exemplars. Data about the student's name, situation and course, besides his code representation in the system will be recorded. 14. … 16. The system should provide information about the recorded students, as well as his personal data such as name, situation, course and code. **Concern: Borrowing** 17. The system should record the borrowing of a book exemplar. The code of the student who will borrow the book will be informed. The data related to the other books already borrowed by the student should be shown. The book is identified by its register number and the current date as well as the date of the borrowing is shown and they may be changed by the library clerk.

[5] Group of requirements that has functional or non-functional interdependency.

[6] the year is converted to 1900 plus year typed

continue

18. The system should record the return of the book. The code of the student who is returning back the book is provided and the system should show the exemplar borrowed by him/her. The book register number is provided to the system, which shows the current date as well as the book return date and they may be changed by the library clerk. The insertion of a note is also allowed.
19. …
21. The system should allow the printing of a receipt at the book return date.

Non-functional Requirements

Concern: Security Access

22. The system should have access restrictions for two groups of users (student and administrator). The group defined as "student" should have permission only to read and consult the collection. The group defined as "administrator" should have access to read and edit all functions.

Concern: Performance

23. The system should accomplish the borrowing operation at a speed of 3.000 milliseconds for each data processed per register.

Concern: Security Copy

24. The system should run a backup on a daily basis, at the end of the working day.

No business rule has been identified during the legacy system execution in the activity *"Develop a use case diagram and document the test cases"*.

Several iterations have been executed in the activity *"Develop the system class diagram"* to obtain the final version of the corresponding artifact (Fig. 2). GRN patterns: one-Identify the Resource, two-Quantify the Resource, and four-Rent the Resource have been identified to compose the diagram. Their respective classes are shown in the upper part of Fig. 2.

The software engineer has used the patterns *script*, created and available in the Rational Rose tool [21], to support the preparation of the artifact *"System class diagram"*. The system classes are represented as subclasses of the patterns classes, with the attributes and methods specific of the system, and are documented in the activity *"Document the changes done in the class diagram"*. Patterns functionalities (methods, relationships, classes, and attributes), used in the activity *"Develop the system class diagram",* but not necessary to represent the legacy system, are also documented in this activity.

UML notes shown in the artifact *"System class diagram"* (Fig. 2) are used to indicate the role of the pattern classes, when necessary, and the implementation of the business rules, when they exist. Some types of data (`DiscreteList`, `TableList`, and `MultivalueList`), specific of the GREN framework, have been used in the artifact *"System class diagram"* to represent some data types not supported by the pattern language.

The activity *"Write the system user's manual"* has not been considered in the agile reverse engineering, because it deals with the preparation of the user manual for the new system implemented, which does not occur when the objective is to conduct the reverse engineering.

Next, in the Construction phase, the artifact *"OO System prototype"* has been created (Fig. 4) in the activity *"Develop an OO system prototype",* to validate the artifact *"System class diagram"* and to elicit new requirements and identify/refine business rules not previously identified. The artifact *"OO System prototype"* has been created with support of the instantiation tool GREN-Wizard [3].

Following, the first iteration of the activity *"Execute the test cases in the OO system prototype"* has been performed. In this activity, test cases documented in the artifact *"Test case documentation"* have been executed on the prototype. For this, the worker followed the execution order of the artifact *"Data dependency diagram among the use cases"*.

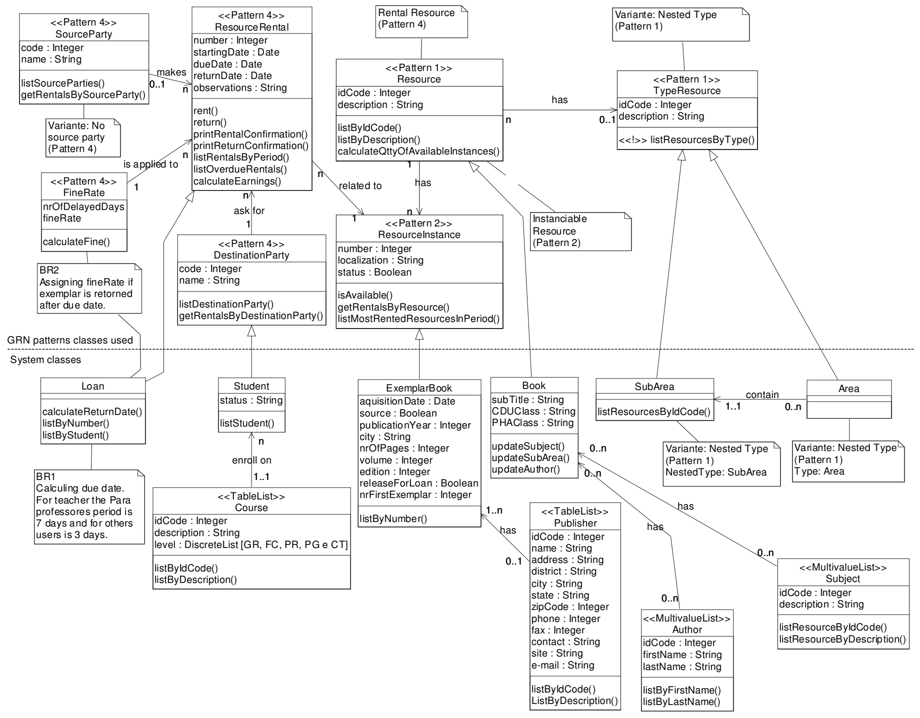


Fig. 2. Library system class diagram

Business rules were not found in the activity *"Execute the test cases in the OO system prototype"*, because the worker observed that they were not implemented on the legacy system. But, the user requested implementation of a business rule (BR1) related to the period authorized for the user to stay with the book. This period is different for teachers (seven days) and other users (three days). This business rule has been documented in the activity *"Document the system business rules"* (Elaboration phase) and the artifact *"Business rules documentation"* has been created. This artifact facilitates the understanding and the implementation of the business rule in a future maintenance activity or when the final objective is the legacy system reengineering. The artifact *"System class diagram"* has also been updated with the `calculateReturnDate` method on `Loan` class (Fig. 2).

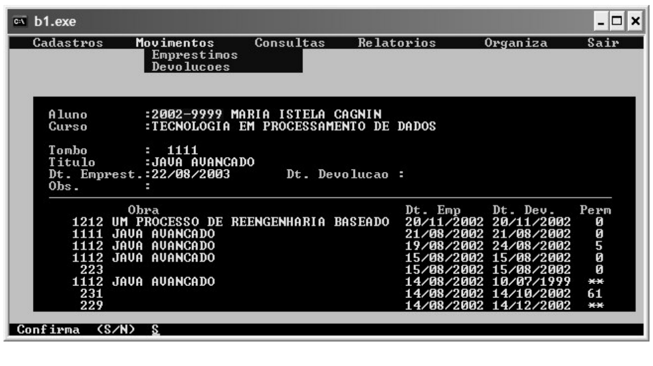
Functionalities that did not belong to the legacy system were identified in the activity *"Execute the test cases in the OO system prototype"* and have been documented. Each functionality was analyzed by the user and he decided to maintain it or not in the system. Furthermore, the worker observed several data consistencies on the legacy system not present on the prototype. This was documented in the activity *"Document the changes done in the class diagram"*.

A demonstration of the OO system prototype has been done for the user, running it concomitantly with the legacy system. In this moment, the user requested the implementation of another business rule (BR2) related to the charge of a fine rate when the library user returns the book after the due date. This business rule has been documented in the activity *"Document the system business rules"*. The artifact *"System class diagram"* has been updated again to represent the business rule implementation. For this, the class `FineRate`, which is a variant participant of pattern 4, has been considered in the artifact *"System class diagram"*.

The activities *"Adapt the OO system prototype", "Convert the legacy system database", "Test the OO system"* and *"Train the end users"* have also not been considered in the reverse engineering, as they are intrinsically related to the system reengineering.

After the completion of each activity iteration, an inspection on the artifacts produced was conducted, according to the verifications previously established on the PARFAIT documentation. All the artifact versions were submitted to the VersionWeb [9] configuration control system.

In Fig. 3, the form for book loan of the legacy system is presented and in Fig. 4 the form of the artifact *"OO system prototype"*, created by the instantiation of the GREN framework in the activity *"Develop an OO system prototype"*, is shown. At the bottom part of the legacy system form, a list with all the books loaned to the student that is doing the loan, is presented. In the artifact *"OO system prototype"*, this list is obtained through the button *List* located at the form upper part. The functionalities of the prototype form bottom part (*Preço total, Desconto total and Total final*) do not belong to the legacy system domain, but were inherited from the framework. When the aim is to conduct only reverse engineering, that is not considered, as in this case the objective is to use the prototype just to elicit new requirements and identify/refine business rules and functionalities specific of the system.

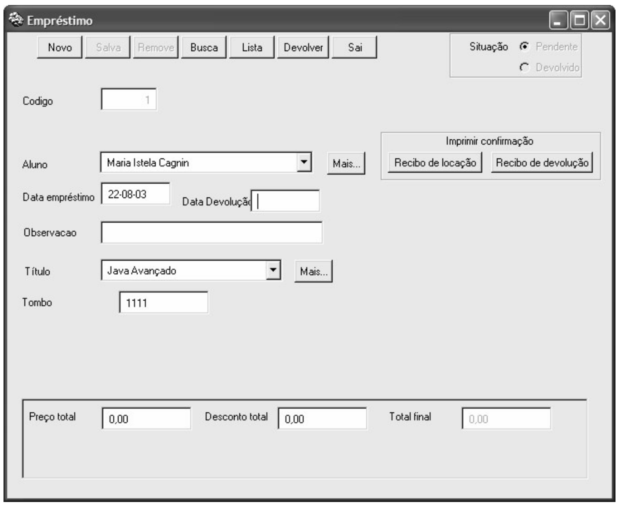


```
Aluno        :2002-9999 MARIA ISTELA CAGNIN
Curso        :TECNOLOGIA EM PROCESSAMENTO DE DADOS

Tombo        :  1111
Titulo       :JAVA AVANCADO
Dt. Emprest.:22/08/2003       Dt. Devolucao :
Obs.         :

           Obra                                 Dt. Emp    Dt. Dev.   Perm
     1212 UM PROCESSO DE REENGENHARIA BASEADO  20/11/2002 20/11/2002   0
     1111 JAVA AVANCADO                        21/08/2002 21/08/2002   0
     1112 JAVA AVANCADO                        19/08/2002 24/08/2002   5
     1112 JAVA AVANCADO                        15/08/2002 15/08/2002   0
      223                                      15/08/2002 15/08/2002   0
     1112 JAVA AVANCADO                        14/08/2002 10/07/1999  **
      231                                      14/08/2002 14/10/2002  61
      229                                      14/08/2002 14/12/2002  **
Confirma  (S/N)  S
```

Fig. 3. Legacy system form for book loan



Situação [✓ Pendente] Devolvido

Codigo	1
Aluno	Maria Istela Cagnin
Data empréstimo	22-08-03
Data Devolução	
Observacao	
Título	Java Avançado
Tombo	1111
Preço total	0,00
Desconto total	0,00
Total final	0,00

Fig. 4. System prototype form for book loan

• PARFAIT/RE

At the beginning, the software engineer decides if the framework may be used, by observing the restrictions and functionalities inherent to its domain. For this, it is necessary to perform the activities *"Obtain familiarity with the framework domain"* (it is not mandatory if the worker has already familiarity with it) and *"Observe the legacy system domain in relation to the framework domain"*.

At least a worker of each role (analyst, programmer, tester, database administrator, etc), that will participate in the agile reverse engineering, must have knowledge about the framework domain. For this, in the activity *"Obtain familiarity with the framework domain"*, it is necessary to read the framework documentation and to run systems resulting of its instantiations. A questionnaire, provided by PARFAIT, must be answered by workers to evaluate their knowledge about the framework domain and then follow the process application guidelines.

In the activity *"Observe the legacy system domain in relation to the framework domain"*, the legacy system is executed in order to observe its features, but the worker must not worry with features details.

If the framework may be used to support reverse engineering, then the elaboration of the artifact *"Requirements Document"* is started and the system general objectives are defined. Test cases and test tools used on legacy system development are recovered to be used, if they exist, aiming at the reduction of reverse engineering time.

Next, in the Elaboration phase, the legacy system is executed to provide familiarity with details of its features and to document them on a use case diagram (activity *"Develop an use case diagram and document the test cases"*). Each feature is considered as a functional requirement and is documented, by concerns, in the artifact

"Requirements Document", and as a use case in the artifact *"Use case diagram"*. The use case description is written based on the feature details, according to the knowledge obtained from its execution. The worker uses input data to run each system feature and obtains the respective output data. These compose each test case that is documented in the artifact *"Test case documentation"*. Other test cases are created from test requirements established by the Equivalence Classes Partitioning and Boundary Value Analysis functional test criteria.

The non-functional requirements are elicited from interviews and questionnaires applied to users and they are also documented in the artifact *"Requirements Document"*.

During the legacy system execution, business rules, if they exist, are identified and documented in an appropriate document named *"System business rules documentation"*. These business rules must be represented as methods/classes/relationships/attributes in the artifact *"System class diagram"* that will be created afterwards.

After elaborating the artifacts *"Use case diagram"* and *"Requirements Document"*, it is necessary to submit it to validation by the legacy system users, who may request new requirements inherent to the organization or the removal of others.

The artifact *"System class diagram"* is produced in the activity *"Develop the system class diagram"* from the GRN pattern language and from the artifact *"Use case diagram"*. For this, the worker must identify which GRN patterns must be used, observing chunks of the legacy system documentation that belong to the GRN domain. After each pattern is identified, the worker reuses the structure section of the pattern documentation (that is, classes, relationships, attributes, and operations, that represent the solution proposed by the pattern) to support the artifact *"System class diagram"* elaboration.

The *script* available on the Rational Rose tool [21] containing all classes, relationships, attributes and operations that represent each GRN pattern structure, can be used to facilitate and to speed up the elaboration of the artifact *"System class diagram"*.

It is stressed here that there may be legacy system functional requirements that cannot be represented by the pattern language and, consequently, will not be obtained from the GREN framework.

Another point is that pattern language patterns may represent functional requirements not included in the legacy system. The software engineer has to consult the users whether they should or not be kept in the documentation that is being made. This is a way to elicit new requirements that belong to the system domain and may be useful for the organization management.

In both cases, the functional requirements are represented in the artifact *"System class diagram"* and documented in the activity *"Document the changes done in the class diagram"*.

Then, in the Construction phase, a first prototype is generated in the activity *"Develop an OO system prototype"*.

The documented tests are applied on the prototype (activity *"Execute the test cases in the OO system prototype"*), so that its behavior may be compared to that of the legacy system, in order to refine or identify new business rules and new specific functional requirements, not previously identified. This is done as follows:

a) each result is compared to the one obtained by the documented test case;
b) if the result is different from the one expected, it is necessary to update the artifact *"Use case diagram"* (if the result represents a new identified or refined requirement), as well as the artifact *"System class diagram"*. In this case, methods, that represent the identified or refined functional requirements/business rules, should be added to the system classes;
c) if the result is the same as expected, it is not necessary to update the diagrams.

The artifact *"OO system prototype*" is also used to elicit new requirements. This is done during prototype demonstration to users, which it is executed concomitantly with the legacy system. If the new requirements suggested by the users are equivalent to patterns of GRN pattern language then a new system prototype is generated.

It is important to mention that all activities described in this section are performed in an incremental way and most of the artifacts elaborated are validated by users.

These guidelines identify a set of activities that, in fact, characterizes an agile reverse engineering based on framework, in agreement with the core principles of Agile Modeling: 1) Software is the primary goal (the goal is to produce software and not documentation useless), 2) Enabling the next effort is the secondary goal (to create enough documentation so that the software evolution can be effective on the next version), 3) Travel light (to create just enough models and documentation to get by), 4) Assume simplicity (to assume that the simplest solutions is the best solution), 5) Embrace change (to accept the fact that change happens and requirements can change during project), 6) Incremental change (to embrace change through incremental approach), 7) Multiple models (each artifacts is appropriate for some situations and not others, then it is necessary to use multiple models to describe software systems), 8) Quality work (to invest the effort to make permanent artifacts of sufficient quality); 9) Rapid feedback (to work closely with customers to understand their requirements and validated them).

5 Conclusions and Suggestions for Future Works

The results of an iterations analyze of activities done in a reengineering prospective case study of a medium size system, using the PARFAIT agile reengineering process, were used to abstract an agile reverse engineering process, referred as PARFAIT/RE. These results have been positive, but will be validated in a case study specific of reverse engineering using the process abstracted, that will be planned and conducted in the near future.

From the reengineering prospective case study conducted, it has been observed that the documentation produced was sufficient to allow the legacy system understanding and was efficient to support the reengineering. We infer that the documentation produced would also be efficient to support maintenance activities on the legacy system. This will be verified in a future case study.

It has been observed that the framework usage, based on the pattern language, promotes reuse of analysis information and provides a system prototype, as fast as possible. It is inferred that this may decrease the time spent in reverse engineering.

With this kind of language, it was possible to obtain the legacy OO documentation, reusing the classes belonging to the patterns structure used.

None of reverse engineering traditional approaches, that the authors are aware of, bear on user participation during reverse engineering as it occurs in PARFAIT/RE. Furthermore, they do not use neither an analysis pattern language to support the creation of the legacy system OO documentation, nor a prototyping paradigm based on frameworks and not even functional tests, to support the requirements elicitation.

The patterns of the analysis pattern language represent requirements that belong to a specific domain. Then, the patterns may bring new requirements not present in the legacy system, but that are interesting for the corporation. On the other hand, the legacy system requirements, not present in the pattern language, can collaborate for its evolution in order to support the requirements elicitation of a larger number of systems. Therefore, the use of pattern languages is an efficient way to support the requirements elicitation both for new systems development and for reverse engineering of legacy systems.

The test cases that have been used to run the legacy system are executed on the prototype, created from the framework instantiation, to identify new requirements and business rules. The system prototype can also be used to elicit new requirements through the prototype demonstration to users, which is executed concomitantly with the legacy system. If the new requirements suggested by the users are compatible to the patterns of GRN pattern language then a new system prototype is generated. Thus, the OO system prototype has also been used as an efficient way to support the requirements elicitation.

It has been observed, with the case study conducted, that a large amount of the effort spent is related to VV&T (Verification, Validation and Test) activities. A way to reduce VV&T time and costs is to associate test cases with each pattern of the pattern language. Thus, it would be possible for the software engineer to reuse the test information both when PARFAIT is used for reengineering as well as when PARFAIT/RE is used to conduct reverse engineering, without compromising the quality of the artifacts produced.

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.
2. Beck, K.; Cunningham, W. *Using pattern languages for object-oriented programs*. Technical Report n. CR-87-43, 1987.
3. Braga, 2003. *A Process for Construction and Instantiation of Frameworks based on a Domain-Specific Patterns Language*. Sc.D. Thesis, ICMC-USP, São Carlos-SP, Brazil, 224 p. (in portuguese).
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, Monticello, Illinois, EUA, v.7, p. 1-33, August, 1999.
5. Ambler, S. W.; Jeffries, R. (2002). *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*. John Wiley & Sons; 1st edition, March.

6. Cagnin, M.I.; Maldonado, J.C.; Penteado, R.; Germano, F. *PARFAIT: Towards a Framework-based Agile Reengineering Process.* In: Agile Development Conference, Salt Lake City, Utha, EUA, 25-29 June, 2003.
7. Cagnin, M.I.; Maldonado, J.C.; Penteado, R.; Germano, F. *PARFAIT: Definition and Application Example.* Working Document, ICMC-USP, São Carlos-SP, Brazil, February, 2003 (in portuguese).
8. Chan, A.; Cagnin, M.I. *Application of PARFAIT Agile Reengineering Process in a Library Control Legacy System.* Working Document, ICMC-USP*,* May, 2003 (in portuguese).
9. Soares, M.D; Fortes, R.P.M.; Moreira, D. A. VersionWeb: *A Tool for Helping Web Pages Version Control*. Proceedings of the International Conference on Internet Mutimedia. Systems and Applications, pp. 275-280, Las Vegas, USA, 2000.
10. Pressman, R. S. *Software Engineering: A Practitioner's Approach*. Publisher: McGraw-Hill, Fifth edition, 2001.
11. Penteado, R. A. D. (1996). *A Method for Object Oriented Reverse Engineering*. Sc.D. Thesis – Instituto de Física de São Carlos-USP, São Carlos-SP, Brazil, 237 p. (in portuguese).
12. Kruchten, P. *The Rational Unified Process: An Introduction*. Second Edition, Addison-Wesley, 2000.
13. Costa, R. M. (1997). *Fusion-RE/I: A Reverse Engineering Method to Support Software Maintenance*. M.Sc. Dissertation. ICMC-USP, São Carlos-SP, Brazil, 112 p. (in portuguese).
14. 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.
15. Taligent Inc. *Building Object-Oriented Frameworks*. URL:http://www-106.ibm.com/developerworks/java/library/oobuilding/?dwzone=java. Accessed: February 2002.
16. Myers, G. J. *The art of software testing*, Wiley, 1979.
17. MySQL (2002). MySQL Reference Manual for Version 3.23. URL: http://web.mysql.com/. Acessed: February, 2002.
18. Turk, D.; France, R.; Rumpe, B. *Limitations of Agile Software Processes*. In: 3rd International Conference on Extreme Programming and Agile Processes in Software Engineering, Alghero, Sardinia, Italy, p. 43-46, May, 2002.
19. Travassos, G.; Shull, F.; Fredericks, M.; Basili, V. (1999). *Detecting Defects in Object-Oriented Designs: Using Reading Techniques to Increase Software Quality.* In: Conference on Object-Oriented Programming, Systems, Languages, and Applications, Denver, Colorado, November.
20. Rajlich, V.; Wilde, N. *The Role of Concepts in Program Comprehension*. In: Proceedings of the 10th International Workshop on Program Comprehension.
21. Rational Rose Corporation. URL: www.rational.com. Acessed: January, 2003.