Software Process Improvement for Small and Medium Enterprises: Techniques and Case Studies

Hanna Oktaba
Nacional Autonomous University of Mexico, Mexico

Mario Piattini
University of Castilla-La Mancha, Spain

Information Science REFERENCE

INFORMATION SCIENCE REFERENCE
Hershey · New York

Acquisitions Editor:	Kristin Klinger
Development Editor:	Kristin Roth
Senior Managing Editor:	Jennifer Neidig
Managing Editor:	Jamie Snavely
Assistant Managing Editor:	Carole Coulson
Copy Editor:	April Schmidt
Typesetter:	Cindy Consonery
Cover Design:	Lisa Tosheff
Printed at:	Yurchak Printing Inc.

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue, Suite 200
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

and in the United Kingdom by
Information Science Reference (an imprint of IGI Global)
3 Henrietta Street
Covent Garden
London WC2E 8LU
Tel: 44 20 7240 0856
Fax: 44 20 7379 0609
Web site: http://www.eurospanonline.com

Library of Congress Cataloging-in-Publication Data

Software process improvement for small and medium enterprises : techniques and case studies / Hanna Oktaba and Mario Piattini, editors.
p. cm.
Summary: "This book offers practical and useful guidelines, models, and techniques for improving software processes and products for small and medium enterprises, utilizing the authoritative, demonstrative tools of case studies and lessons learned to provide academics, scholars, and practitioners with an invaluable research source,"--Provided by publisher.
ISBN 978-1-59904-906-9 (hbk.) -- ISBN 978-1-59904-908-3 (e-book)
1. Computer software--Development--Management. 2. Computer software industry--Management--Case studies. 3. Small business--Data processing--Case studies. I. Oktaba, Hanna. II. Piattini, Mario, 1966-
QA76.76.D47S66354 2008
005.1--dc22
2008001869

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter XIII
An Incremental Functionality-Oriented Free Software Development Methodology

Oswaldo Terán
ENDITEL; Centro de Micro Electrónica y Sistemas Distribuidos and Centro de Simulación y Modelos, Universidad de los Andes, Venezuela

Johanna Alvarez
CENDITEL, Venezuela

Blanca Abraham
CEMISID Universidad de los Andes, Venezuela

Jose Aguilar
CENDITEL; Centro de Micro Electrónica y Sistemas Distribuidos, Universidad de los Andes, Venezuela

ABSTRACT

This chapter presents a methodology used as reference model for a free software factory that is part of the National Centre for Free Technologies in Venezuela. This centre is oriented at promoting free software development for serving mostly the public sector in order to promote endogenous development and technologic autonomy. Under this strategy, strengthening the software small and medium size enterprises and cooperatives, by allowing them to participate in different projects (improving their know-how) and providing them with a methodology for increasing their capabilities and software quality, is necessary and urgent. This methodology plans the development process incrementally, based on a prioritisation of *the software functionalities development in accordance to the functionalities risks, development urgency, and dependencies. It combines aspects of the two styles of free software development, namely cathedral and bazaar. The development process is centralised, in essence collaborative, and continuously allows source code release.*

INTRODUCTION

The Free Software Factory (FSF) of CENDITEL (Venezuelan national centre for promoting free technologies) has been conceived and created as part of the efforts of the Venezuelan State aiming at increasing endogenous development and technological sovereignty. In particular, it intends to strengthen the national software sector, especially the small and medium software enterprises (including the cooperatives), by allowing them to access the technology and participate in the software market, on one hand, and to increase their capabilities and software quality, on the other hand.

Two styles exist for developing free software: the cathedral style and the bazaar style. In the cathedral mode, software is developed from a unified *a priori* project that prescribes all the functions and the features to be incorporated in the final product. Programmers' work is centrally coordinated and supervised in order to assure the integration of various components. On the other hand, in the bazaar style, software emerges from an unstructured evolutionary process. Starting from a minimal code, groups of programmers add features and introduce modifications and patches to the code. There is no central allocation of different tasks; developers are free to develop a given program in directions they favor.

This chapter presents an attempt at building a free software development methodology having many characteristics of the cathedral style but keeping certain principle of the bazaar mode. The methodology has been developed at a public organisation which responds to public sector free software necessities and requirements that must be satisfied in a limited time period. Because of this, it is necessary to adopt the cathedral mode of work while taking key advantages of the bazaar style. For instance, it is allowed that developers from outside the organisation contributes with software coding, testing, and so forth; these external developers do not follow a centrally controlled process; and the software code is made public as soon as it is tested.

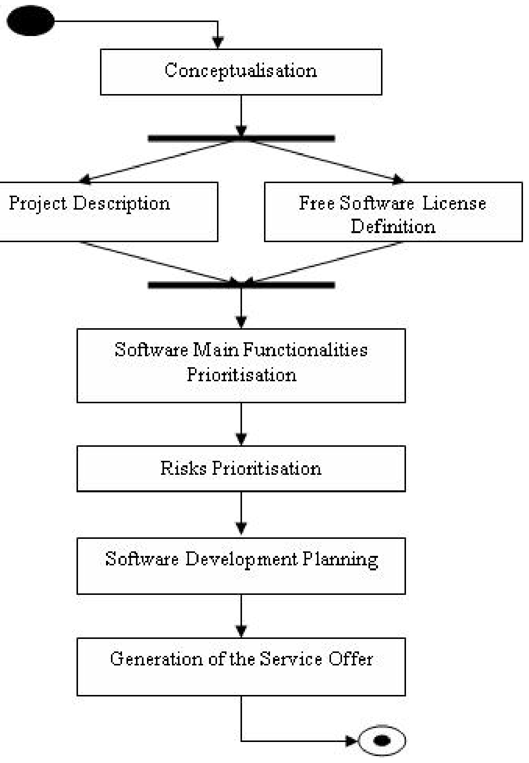
This methodology assumes an organisational structure oriented towards specific processes. The processes dedicated to software development are:

- *Process # 1*: Free Software Project Management
- *Process # 2*: Specific Project Administration
- *Process # 3*: Free Software Application Development

Actions to be carried out in these processes are classified in steps and activities. In particular, steps and activities in the third process are implemented by the following six phases. This methodology has taken ideas from diverse software development methodologies, methods, and models such as the extreme programming method (Beck, 2004), the rational unified process (Kruchten, 2000; Pollice, 2001; Probasco, 2000), the watch method (Montilva, 2004; Montilva, Hamzan, & Ghatrawi, 2000), and the model of processes for software development (MoProSoft) (Oktaba et al., 2005). Due to the fact that these models and methods, except extreme programming, have been proposed proprietary software development, it has been necessary to adapt the hints, ideas, or procedures taken from them to the free software development needs.

The methodology to be proposed has been validated at the FSF of the Foundation for Science and Technology of the Mérida State in Venezuela (FUNDACITE-Mérida). This factory has permitted us to understand better, empirically, the real needs of a free software development process and has also been a source of interesting ideas. The proposed structure will allow planning and control activities which are required in the management and administration of software projects. In addition, the free software application development process is iterative and incremental, in terms of the software application functionalities. On the other hand, the design of the application is based on component architecture, which allows software reuse. Each process will be explained in detail in the main body of this chapter. For facilitating each process, some free software tools will be suggested.

Figure 1. Free software project management process


Conceptualisation
Project Description
Free Software License Definition
Software Main Functionalities Prioritisation
Risks Prioritisation
Software Development Planning
Generation of the Service Offer

FREE SOFTWARE DEVELOPMENT METHODOLOGY

Process # 1: Free Software Project Management (FSPM)

This process is responsible for managing all projects being carried out by the organisation, that is, both internal projects (projects for the organisation) and external projects (projects for other organisations) are managed. Specifically, in this process, a "service offer" for the project to be developed is generated. This offer must include a conceptualisation, a description, and a (general) development plan for the project, as well as the definition of the free software license. Figure 1 shows the steps as a workflow for the FSMP process. Subsequently, these steps will be described.

a. **Conceptualisation**
Description: Specific user needs and/or problems must be identified in order to define the scope of the project. In case of a high complexity of the project, for instance, the scope of the problem goes beyond a software need and involves organisational issues, methodologies such as technologic prospective and strategic planning, and/or tools such as fish bone (a technique commonly used in operation research and in quality control) are recommended. In this case, the study would suggest a set of solutions and organisational changes, of which software needs would be only part of the whole answer.
People Responsible: According to free software philosophy, all people associated with the project (project manager, project administrator, developers, users, and so forth) must

be involved from the very early stages of it, even when new actors can be incorporated to any phase of the project.
Techniques: Prospective analysis*,* strategic planning, or any other useful technique.
Products: (1) Client/user's needs and/or problems; (2) scope of the project.

b. **Project Description**
The main point in this phase is to achieve a detailed description of the project. Each actor must analyse the client's need and problem, as well as the project's scope, in order to contribute to the description of the project.
People Responsible: Clients, users, project manager, project administrator, developers, assessors and other people interested in the project participation.
Products: Project description document.

c. **Free Software License Definition**
Description: In this step, the free software license to be adopted for the development is defined. It might be the case that a license from the market satisfies the client requirements and then is chosen or, in case that there is not an existent license matching the user's requirements, a new license is defined.
People Responsible: Clients, users, project manager, project administrator, assessors.
Products: Project license.

d. **Software Main Functionalities Prioritisation**
Description: In this step, the goal is to describe and classify the software functionalities in accordance with the implementation urgency required by the client.
People Responsible: Clients.
Products: Functionalities prioritised.

e. **Risk Prioritisation**
Description: To identify, prioritise, and associate the risks to software application functionalities. The risks are prioritised in accordance to their impact on the application development.
People Responsible: Clients, users, project manager, project administrator, developers, assessors, and other people interested in the project.
Products: Risks prioritised.

f. **Software Development Plan**
Description: To build the development plan, the implementation order of the functionalities of the application must be established, in accordance to the functionality priorities defined by the client, and the risks prioritised associated to the functionalities. This must allow determining the number of cycles or iterations required for the development of the application. A cycle is responsible for developing a certain number of functionalities (taken into account their priority order). A development plan can be modified after an iteration, as a result of work reorganisation, in line with the dynamic of the project.
People Responsible: Project manager, project administrator, and developers.
Products: Development plan.

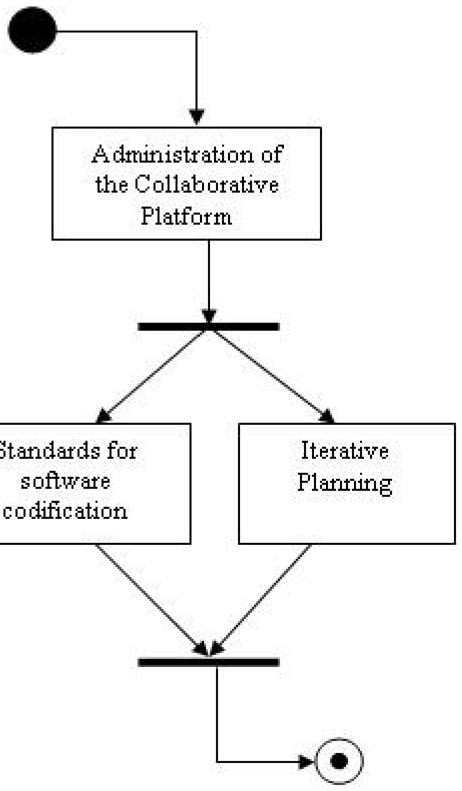
g. **Generation of the Service Offer**
Description: The service offer is completed in this step. It specifies all important issues of the project, such as the goal, scope, and description of the project; the development plan; the due dates for deliverables; the work team; the project cost; and the operation platform.
People Responsible: Project manager, clients.
Techniques and Tools: Service offer forms.

Note: In accordance to the chosen software license, the products achieved in this process must be published in a collaborative development platform. This will facilitate the interested people access to the software products and their documentation.

Process #2: Administration of Specific Projects (ASP)

The administration of specific projects leads the developer team of a software application (it is assumed that a software application development corresponds to a software project). In this sense, each software project must have at least one project administrator. The project administrator is responsible for organising and planning the activities corresponding to each iteration defined in the development plan. Additionally, the project administrator must assure the software quality, manage the system configuration, and the collaborative technical platform, as well as supervise and control the project development and the administration of subcontracts. Figure 2 shows the main steps as a workflow for the ASP process. Subsequently, these steps will be described.

Figure 2. Process for the Administration of Specific Projects


Administration of the Collaborative Platform
Standards for software codification
Iterative Planning

a. **Administration of the Development Collaborative Platform**
Description: The processes related to the ASP phase are facilitated by using a collaborative platform. A software developing team has its own necessities; accordingly it is important to select the correct tool for collaborative software management considering such necessities. In this phase, the collaborative platform is set up. The collaborative platform permits any interested person to collaborate with and share ideas, source code, documentation, testing, and so forth. This is a very important aspect of the free software development. However, as part of the administration of this platform, the project administrator must approve and then publish the software versions and the associated documentation in accordance with the free software license assumed.
People Responsible: Project administrator.
Tools: GFORCE, and so forth.
Products: Collaborative development server.

b. **Standard for Software Codification**
Description: This step establishes the standards for code generation and for the documentation to be used during the software development. These standards allow a quick and simple reading of the code, facilitating the work of the whole group, including the client, the user, and other actors.
People Responsible: Project administrator.
Products: Coding and documentation standards.

c. **Iterative Planning**
Description: The activities of the iteration to be carried out in accordance with the functionalities to be developed are planned. After an iteration is performed, the next iteration is planned and takes into account functionalities that could not be implemented and problems found during the previous iteration.
People Responsible: Project administrator.
Techniques: Gantt, Pert/CPM

Tools: Planner, GFORCE, XPTracker, SourceForge, and so forth.
Products: Plan for the iteration to be developed.

Note: Products accomplished in this process must be placed at the collaborative platform in accordance with the adopted free software license.

Process #3: Free Software Application Development

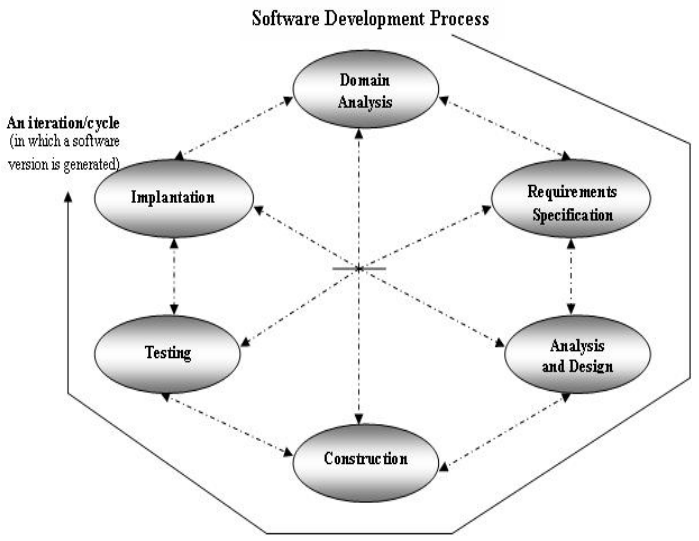
The software application is constructed by the sequence of iterations or cycles in an incremental and iterative way, allowing that users and clients can check the advances of the work and give feedback useful for improving the development and testing processes. The methodology presents a general reference framework or structure for the activities to be planned at each iteration of this process (see Figure 3). In each cycle, one activity receives the main attention while the others are secondary. The whole set of functionalities is developed during the cycles.

Any person can access and execute the source code stored in the collaborative platform. In this way, everyone can contribute to the project. Experiences show that the more people use and test the software, the more quickly the errors and bugs will be found and solved.

It is important to mention that during the software application construction, not only must the code be published but also all associated documentation. In this manner, new programmers and collaborators can be easily incorporated. As mentioned earlier, the code and associated documentation publication depends on the software license established. Next, in Figure 3, the development phases carried out during this process will be presented.

a. **Application Domain Analysis**
Description: This phase is considered one of the most important in the software development process, since the domain environment and context where the application will operate are analysed and understood. Such analysis is carried out in the first iteration but can be upgraded in the subsequent iterations. The

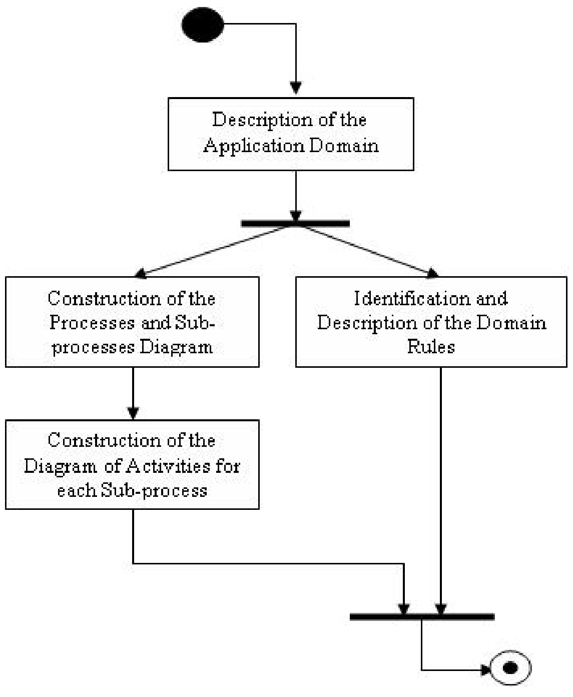
Figure 3. Free software development process


Software Development Process
An iteration/cycle
(in which a software version is generated)
Domain Analysis
Requirements Specification
Analysis and Design
Construction
Testing
Implantation

activities workflow for this phase is shown in Figure 4. Following this figure, the main activities will be described.

- ***Domain Description***
  Description: Establishes and validates the application domain and its organisational scope.
  People Responsible: System analyst internal to the developer organisation, users, clients, programming team. In this chapter, the phrase "*internal to the organization*" means a person who works for the organization that develops the software, as opposed to "external to the organization," which means a person who is not actually working for the developer organization.
  Techniques: Domain engineering, interviews, revision of documents, and bibliography.
  Products: Domain application definition.

Figure 4. Steps for the application domain analysis phase


Description of the Application Domain
Construction of the Processes and Sub-processes Diagram
Identification and Description of the Domain Rules
Construction of the Diagram of Activities for each Sub-process

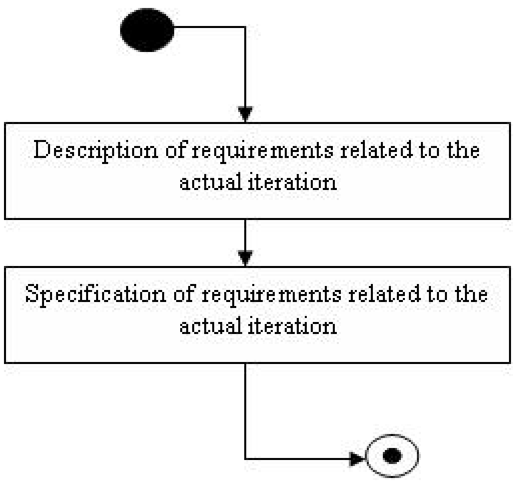
- ***Construction of the Processes and Subprocesses Diagram***
  Description: This activity must identify processes and subprocesses related to the application domain, as well as events associated to all these, in order to generate the domain processes and subprocesses diagram. Finally, this diagram must be validated.
  People Responsible: System analyst (internal to the organisation), users, clients.
  Techniques: The processes diagram given by UML.
  Tools: Umbrello, ArgoUML, caseUML.
  Products: Domain processes and subprocess diagram.
- ***Construction of the Diagram of Activities for Each Subprocess***
  Description: Generates and validates the activities diagram for each subprocess.
  People Responsible: System analyst (internal to the organisation), users, clients.
  Techniques: Activities diagram offered by UML.
  Tools: Umbrello, ArgoUML, caseUML.
  *Products: S*ubprocess activity diagrams.
- ***Identification and Description of the Domain Rules***
  Description: Domain rules regulating the application domain must be identified and studied.
  People Responsible: System analyst (internal to the organisation), users, clients.
  Techniques: the activities diagram offered by UML.
  Tools: Umbrello, ArgoUML, caseUML.

*Products: S*ubprocess activity diagrams.

b. **Requirements Specification**
Description: In this phase, the functionalities to be developed in the planned iteration are specified, and the nonfunctional requirements are defined or upgraded. Generally, the non-functional requirements are defined in the early iterations. The requirement specification document will be upgraded from iteration to iteration. It is important to notice that in this phase the user or the client can modify, change, include, or eliminate requirements and risks, which, in turn, might entail updates of the development plan. The activities workflow for this phase is shown in Figure 5. Following this figure, the main activities for this phase will be described.

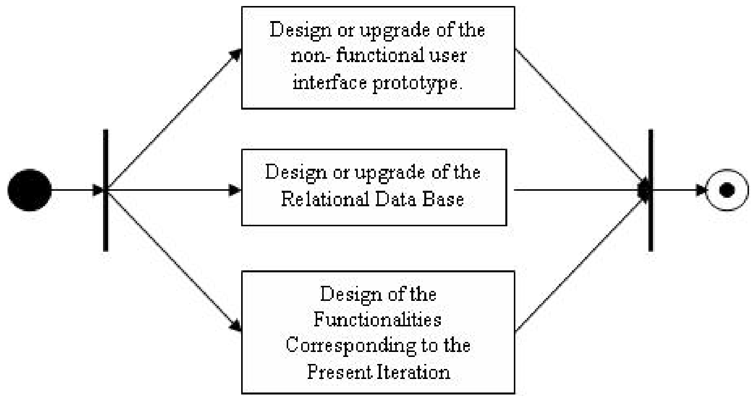
- ***Description of Requirements Related to the Actual Iteration***
  Description: A detailed description of the functional requirements for the present iteration is generated, and the nonfunctional requirements are defined or upgraded. These will allow generating and validating the requirements definition document. It is important to mention that only the definition of such requirements related to the present iteration are validated, since those requirements related to previous iterations were validated during the corresponding iterations.
  People Responsible: System analyst (internal to the organisation), users, clients.
- ***Specification of Requirements Related to the Actual Iteration:***
  Description: To create or upgrade the requirements specification document, including the use cases describing the functional requirements associated with the actual iteration. In this methodology, it is understood that the requirements specification is made in terms of diagrams and textual descriptions of the use cases. Only the specification of those requirements associated to the present or actual iteration must be verified and validated in this step.
  People Responsible: System analyst (internal and/or external to the organisation), users, clients.
  Techniques: the use cases diagram offered by UML.
  Tools: Umbrello, ArgoUML, caseUML.

Figure 5. Steps for the requirements specification


Description of requirements related to the actual iteration
Specification of requirements related to the actual iteration

c. ***Analysis and Design***
Description: In this phase, the specification of requirements is translated into a design specification, based on a set of architectonic views, which represent the system architecture. In this phase also, the user interfaces and databases are designed. The application architecture, like the requirements, is enriched or upgraded as the subsequent iterations are carried out, since each iteration add functionalities to the software application been developed. All this gives flexibility to the design, permitting that change in the client's viewpoint about desired functionalities be taken into account without great difficulties.

Figure 6. Steps for the analysis and design phase


Design or upgrade of the non- functional user interface prototype.
Design or upgrade of the Relational Data Base
Design of the Functionalities Corresponding to the Present Iteration

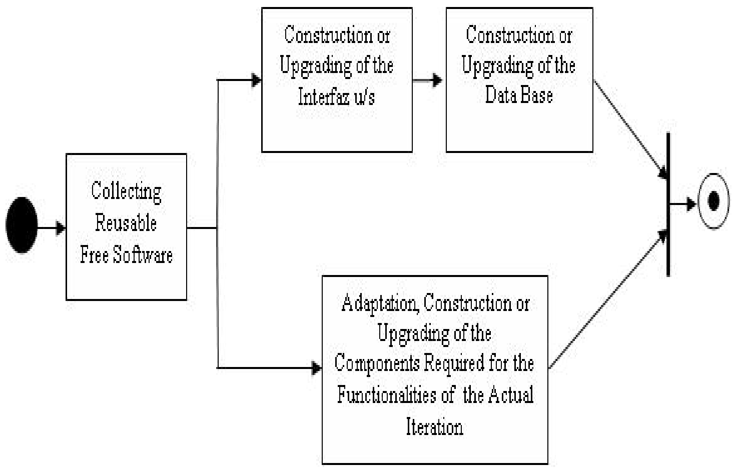
The activities workflow for this phase is shown in Figure 6. Afterwards, the main activities are going to be described.

- ***Design or Upgrade of the Nonfunctional User Interface Prototype***
  Description: Create or update the nonfunctional user interface prototype. This design includes the hierarchic diagram of windows, taking into account the user requirements. This design must be validated.
  People Responsible: System analyst (internal or external to the developer organisation), programmers, users, clients.
  Tools: UX, DIA.
  Products: Design of the nonfunctional user interface prototype.
- ***Design or Upgrade of the Relational Database:***
  Description: The database design document is generated or upgraded. This document must contain the diagram entity/relation and the relational scheme, for the actual iteration. For these diagrams, the entities of the database and their attributes, as well as the primary and the foreign keys, must be defined. The entities of the database are identified by using the use cases for the present iteration. Finally, the database administrative procedures (i.e., backup, security, recovery, etc.) must be defined, and the database design document must be validated.
  People Responsible: System analyst (internal and external to the organisation), programmers.
  Techniques: Normalisation formulas.
  Products: Entity relation diagrams.
- ***Design of the Functionalities Corresponding to the Present Iteration***
  Description: The architectonic views must be generated or upgraded. It is constituted by the logic, the implementation, the behaviour, and the conceptual views. The logic view is defined by the class diagrams of the software application. It is created or upgraded by: (a) deriving from the use cases (associated to the actual iteration) the objects of the application, (b) generating the sequence diagrams for the "methods" or functions involved in the realisation of the use cases for the actual iteration. The implementation view is generated or upgraded from the components diagrams. The behaviour view is created or upgraded from the interaction relations

among the components. The conceptual view is defined or upgraded from the use case diagrams corresponding to the actual iteration.
People Responsible: System analyst (internal or external to the organisation).
Techniques: Class, components, and interaction diagrams
Tools: Umbrello, ArgoUML, caseUML.
Products: Architectonic view of the software application.

d. **Construction**
Description: For the actual iteration or cycle, the user interface, the database, and the functionalities of the application are constructed or upgraded in this phase. For that, the software application source code for the actual version is developed. The activities workflow for this phase is shown in Figure 7. Afterwards, the main activities of this phase will be described.
   - ***Collecting Reusable Free Software*:**
     Description: Reusable free software components, abstract data type, classes, functions, and whole systems, useful for the software application, are searched for and collected.
     People Responsible: Programmers (internal and/or external to the organisation). This is the first activity where external programmers participate in the software development.
     Tools: Some are available at Web sites such as www.fsl.funmrd,gov.ve, freshmeat.net, sourceforge.net, and so forth.
   - ***Construction or Upgrading of the User Interface U/S***
     Description: The reusable user interface components corresponding to the design of the interface associated to the actual iteration are adapted and, when required, those of the previous iterations are upgraded.
     People Responsible: Programmers (internal and/or external to the organisation).
   - ***Construction or Upgrading of the Database***
     *Description***:** Build or upgrade the database using information from the database design document for the actual iteration. Additionally, components of the user interface must be integrated along the database.

Figure 7. Steps for the construction phase

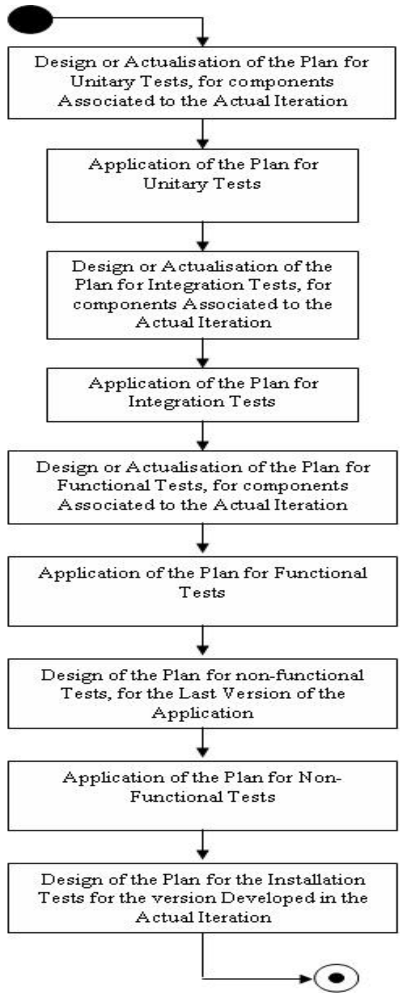

Collecting Reusable Free Software
Construction or Upgrading of the Interfaz u/s
Construction or Upgrading of the Data Base
Adaptation, Construction or Upgrading of the Components Required for the Functionalities of the Actual Iteration

People Responsible: Programmers (internal and/or external to the organisation).
Techniques and Tools: Web sites like www.fsl.funmrd,gov.ve, freshmeat.net, sourceforge.net, and so forth.

- ***Adaptation, Construction, or Upgrading of the Components Required for the Functionalities of the Actual Iteration***
  Description: For the components, abstract data types, classes, and functions requited for the functionalities of the actual iteration: (a) adapt those reusable already collected, (b) construct those that could not be found, (c) update those useful from previous iterations.
  People Responsible: Programmers (internal and/or external to the organisation).
  Tools: Some are available at Web sites such as www.fsl.funmrd,gov.ve, freshmeat.net, sourceforge.net, and so forth.

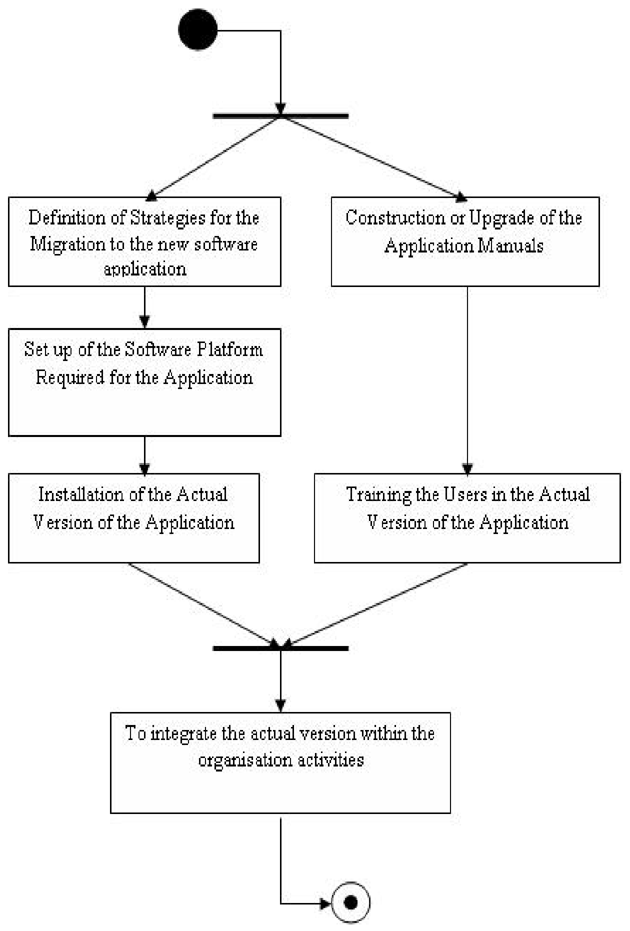
e. **Testing**
Description: In this phase, the unitary, integration, functional, and nonfunctional tests, for the components corresponding to the functionalities associated to the actual iteration, are designed or upgraded, and applied. The nonfunctional tests are applied only in the last version of the software application, which is obtained in the last iteration. The installation tests are also designed in this phase, but are applied in the implantation phase. It is important to say that code modified by developers external to the developer organisation must also be appropriately tested. Only after these tests are successfully completed can the project administrator publish the code. Figure 8 shows the workflow for this phase. Since this figure sufficiently explains each step, in the text following the figure, only the people responsible, techniques, tools, and products for a test design/upgrade or for a test application will be specified.

Figure 8. Steps for the testing phase


Design or Actualisation of the Plan for Unitary Tests, for components Associated to the Actual Iteration
Application of the Plan for Unitary Tests
Design or Actualisation of the Plan for Integration Tests, for components Associated to the Actual Iteration
Application of the Plan for Integration Tests
Design or Actualisation of the Plan for Functional Tests, for components Associated to the Actual Iteration
Application of the Plan for Functional Tests
Design of the Plan for non-functional Tests, for the Last Version of the Application
Application of the Plan for Non-Functional Tests
Design of the Plan for the Installation Tests for the version Developed in the Actual Iteration

- **Test design/upgrade:**
  People Responsible: Tester (internal and/or external to the organisation).
  Techniques: White and black box tests.
  Products: Test plans.
- **Test application:**
  People Responsible: Tester (internal and/or external to the organisation).
  Tools: Test, Check, Junit, Cppunit.
  Products: Test reports.

Figure 9. Steps for the installation phase


Definition of Strategies for the Migration to the new software application
Construction or Upgrade of the Application Manuals
Set up of the Software Platform Required for the Application
Installation of the Actual Version of the Application
Training the Users in the Actual Version of the Application
To integrate the actual version within the organisation activities

f. **Implantation**
In this phase, the actual iteration version is released to the client so that the client can validate this version while other functionalities are developed in the next iterations. The migration strategies towards the new application are defined, the user is trained for using the delivered version, the actual version is installed, the installation test is applied, and the software application manuals are generated or upgraded, and verified. Finally, the actual version of the software application is integrated along with the organisation activities. Figure 9 shows the workflow for this phase. Given that the figure sufficiently explains each step of this phase, a detailed description for each step will not be given. However, it is important to mention that: (a) the people responsible for these steps are programmers and testers (both internal to the developer organisation) and (b) the main products of this phase are the system manuals, the training material, and the installation test reports for each installed version.

STUDY CASE

The presented methodology has been shaped and updated recently and has been implanted partially only in two projects. The implantation process will continue during the present year (2007) in order to apply the whole methodology to all projects. The two projects involved in the implantation process until now are:

1. The Dis-centralised Administrative System.
2. The Automation of the FSF. This means the automation of the free software project management, administration of specific projects, and software application development processes.

The Dis-centralised Administrative System had already been started and was entering the test phase at the moment the methodology began implantation. Because of this, until now, the methodology has been applied in this project only for part of the software application development process, more specifically, for the unitary tests of the test step. On the other hand, since the automation of the FSF is still in course, the methodology has been applied up to the point the project has reached at present. However, the methodology has been applied from the beginning of the project. The following processes of the methodology have been implanted: the free software project management**,** administration of specific projects, and some aspects of the software application development process. Next, details about the application of the methodology to both of these projects will be presented.

Case 1: Automation of the Free Software Factory

Process # 1: Free Software Project Management

a. **Conceptualization**
   Results of carrying out this step are summarised in a set of filled templates, which are stored in the GForge server (see Alvarez, 2007, sections 1 and 2). These templates show needs and problems, and scope of the project of implementation of the FSF. Those problems and needs include:
   - Lack of a database of digitalised templates for documenting the development processes.
   - The dynamics of the demand requires urgent development.
   - Need of a knowledge base for learned lessons.

   The scope of the project delimitates the system to be developed in terms of which processes and which steps will be covered.
b. **Project Description**
   This gives an overview of how the automation of the FSF project is being carried out.
c. **Free Software License Definition**
   There is not any licence defined for this project. All material developed in this project will be available from the GForge server.
d. **Software Main Functionalities Prioritisation.**
   All functionalities to be covered by the automation system of the factory were defined and prioritised. Results of this phase are presented in Alvarez (2007, section 3). Among these functionalities, we have:
   - Digitalise the functionalities and the risks prioritisation templates.
   - Select analysis and design tools.
   - Choose testing tools.
   - Automate the project plan template.
   - Integrate templates and design tools.
   - Integrate design and programming phases (generation of automatic code)
   - Develop a knowledge base.

   These functionalities received a weight, as the methodology states.
e. **Risk Prioritisation**
   At this step, the more important development risks were defined (see Alvarez, 2006, section 4 for more details). Among these risks, we have:

- Scarcity of free automation tools and FSF's development team's lack of knowledge and capacities for building, design, and test tools for free software development.
- Few people dedicated at testing and short experience in testing.
- Low experience in following methodologies.

As above, a weight is associated to these risks.

f. **Development Priorities Definition**
A prioritisation of the functionalities was performed (Alvarez, 2006, section 5), by following this formula:
Total functionality F_1 weight = ($\sum VR_i$ para F_1) * PR + VF_1 * PF , where,
- the VR_i are the risks for the functionality F_1;
- VF_1 is the weight for the functionality F_1;
- PR and PF are the relative weights-of-the-factors, in this case, between the total sum of risk weights, factor 1, and the functionality weight, factor 2.

Following this procedure, one of the most important functionalities resulted to be automated design and testing facilities. Tools for these tasks were selected from those available on the Internet.

g. **Software Development Plan**
This plan presents the development schedule, indicating the functionalities to be developed at each iteration; there were seven defined iterations after considering the functionality dependencies, size of the development team, and the functionalities development prioritisation (for more details, see Alvarez, 2007, section 5).

h. **Generation of the Service Offer**
The service offer indicates (Alvarez, 2007, section 6), for instance:
- The offer proposal: to develop a system to automate the free software development processes.
- The project scope: to automate in some degree, by integrating and digitalising (and in some cases completely automating, when the complexity of these tasks allows it) tools for implementing the FSF.
- The release schedule (to the client).

Process #2: Administration of Specific Projects

a. Administration of the Collaborative Platform.
GForce was installed as the collaborative Platform.
b. Standards for software codification.
The codification standard is being defined at present.
c. Iterative planning.
This step is performed by using a GForce scheduling facility (Alvarez, 2007, section 7).

Process #3: Software Application Development

The software application development process consists basically on programming on top of GForge, in order to adapt it and add functionalities, to permit carrying out the software development activities required by the FSF processes. Some of the adaptations already implemented are:

- Digitalisation of templates, among which we have: client's needs and problems; scope of the project; service offer; test plan; test reports.
- Automation of the project plan.

Case 2: Dis-Centralised Administrative System

Process #3: Software Application Development

As mentioned previously, for this project, the methodology has been implanted to perform the unitary test plan (Alvarez, 2007, section 8).

CONCLUSION

The presented methodology pretends strengthening the software national sector, especially the small and medium software enterprises (including the cooperatives), by allowing them to access the technology and participate in the software market through a collaborative development of software for the public administration (main goal of the FSF), on one hand, and to increase their capabilities and software quality, on the other hand.

In this sense, the development process presents certain specific characteristics and numerous advantages (as it is stated in the methodology). As said before, a fundamental aspect is the collaborative development by iteration: a particular development group may enter or leave to collaborate at any iteration in accordance to the group interests. Additionally, the software developments and upgrades coming from any involved group are open to the community via a collaborative platform. Consequently, the development groups get benefits from a methodological framework, which establishes the ways and moments for participation, forms to recovery versions of the development, development rules and tools, and so forth. All these practices on the bases of the development framework allow any small or medium enterprise to share/communicate with partners in the free software development community, in accordance with the free software development philosophy. The proposed methodology has been partially validated at the FSF of the Foundation for Science and Technology of the Mérida State in Venezuela (FUNDACITE-Mérida). In addition, this factory has permitted to understand better, empirically, the real needs of a free software development process and has also been a source of ideas.

ACKNOWLEDGMENT

This chapter has been developed inside the project: "Process Improvement for Promoting Iberoamerican Software Small and Medium Enterprises Competitiveness – COMPETISOFT" (506AC287) financed by CYTED (Programa Iberoamericano de Ciencia y Tecnología para el Desarrollo)

REFERENCES

Alvarez, J. (2007). *Resumen del avance de la aplicación de la metodología desarrollada para la Fábrica de Software Libre* (Tech. Rep. No. 001-2007). Fundacite, Merida: Fábrica de Software Libre. Retrieved December 17, 2007, from http://www.funmrd.gov.ve/drupal/files/technicalReportJohanna.pdf

Alvarez, J., Aguilar, J., & Teran, O., et al. (2006). *Metodología para el Desarrollo de Software Libre: Buscando el Compromiso entre Funcionalidad y Riesgos* (Tech. Rep. No. 001-2006). Fundacite, Merida, Venezuela: Fábrica de Software Libre.

Beck, K. (2004). *Extreme programming explained: Embrace change* (2nd ed.). Addison-Wesley Professional.

Corredor IIMI. (2006). *Evaluación de MoProSoft como alternativa metodológica de organización de empresas de desarrollo y mantenimiento de software*. Tesis de Pregrado, Escuela de Ingeniería de Sistemas-Universidad de Los Andes, Mérida, Venezuela.

Kruchten, P. (2000). *The rational unified process: An introduction* (2nd ed.). Addison-Wesley.

Montilva, J. (2004). Desarrollo de Aplicaciones Empresariales: El Método WATCH. Mérida, Venezuela: Jonás Montilva.

Montilva, J., Hamzan, K., & Ghatrawi, M. (2000, July). The watch model for developing business software in small and midsize organizatios. In *Proceedings of the IV World Multiconference on Systemics, Cybernetics and Informatics (SCI'2000)*, Orlando, FL.

Oktaba, H., Alquiara, C., Su, A., Martinez, A., Quintarilla, A., Ruvalcaba, M. (2005). *Modelo de Procesos para la Industria de Software* (MoPro-Soft, Versión 1.3). México. Retrieved December 16, 2007, from http://www.software.net.mx

Pollice, G. (2001). *Using the rational unified process for small projects: Expanding upon eXtreme programming* (White Paper TP 183). Rational Software.

Probasco, L. (2000). *The ten essentials of RUP: The essence of an effective development process* (White Paper TP177). Rational Software.