ONTO-RUP: A RUP Based Approach for Developing Ontogenetic Software Systems

Farida Kherissi and Djamel Meslati

LRI Laboratory,
University Badji Mokhtar-Annaba
BP 12, 23000, Annaba, Algeria
Kherissi_farida@hotmail.com, Meslati_djamel@yahoo.com

Abstract. It is impossible to produce systems of any size which do not need to be changed. Once software is put into use, new requirements emerge and existing requirements change as the business running that software changes. Ontogenetic software systems have the ability to evolve dynamically in an autonomous way to meet the user needs and the anticipated and unanticipated changes of requirements. The evolution of these systems has the particularity to be a continuous process that shapes them from the beginning of their creation. This characteristic does not match the current development methods which consider the evolution a sporadic process. All current methods are still unsuitable for development of ontogenetic software systems. Indeed, they do not provide any tool or artifact to take into account the anticipated and unanticipated changes. In this article, we propose an extension of the Rational Unified Process that aims at providing a preliminary framework that allows developing ontogenetic systems.

Keywords: Use Cases, Change Cases, Ontogenetic Systems, Rational Unified Process.

1 Introduction

Many of IT experts talk about evolution modeling and/or developing flexible software [1, 6, 8, 11]. They consider flexibility a good feature and ease of software system modification one of the most important attributes, but they do not say how to achieve it. The knowledge of a system's requirements is necessary imperfect because a significant part of those requirements lies in the future and is unknowable at the time the system is designed. The software engineering community expects that most of the future innovations in information technologies will likely take place in the context of the development methods. These methods have to face the software evolution problems and complexity. Indeed, we are now approaching the limits of our capabilities to deal with complex software systems [17]. Inspired by nature and biology, researchers are now considering alternative solutions to software evolution problems such as ontogenetic software systems, where the main feature is to evolve dynamically and in an autonomous way. Like biological organisms ontogenetic systems

E. Altman et al. (Eds.): Bionetics 2009, LNICST 39, pp. 140–151, 2010.

evolve dynamically and the change mechanisms shaping them throughout their lives are considered as their ontogenetic imension (also called ontogenesis). Taking into account ontogenesis, when developing software systems, is a new challenge and a radical vision of the evolution that will have an influence on our perception of software systems as well as on our approaches of designing them. As an illustrative example, Mage is a bio-inspired approach of ontogenetic systems that is based on genetics [14, 15]. It provides concepts to model the ontogenesis of a software as an embedded genome, whose role consists of shaping continuously this system according to the anticipated and unanticipated changes that may occur. While anticipated changes are those identified before the release of the software system, unanticipated changes emerge when the system is used. According to the Mage approach, a system is composed of two parts a genome and a phenotype. The phenotype is the equivalent of the classical software code. The genome is a collection of genes which continuously shape the phenotype.

The Mage approach needs and advocates devising new suitable development methods in order to benefit from the ontogenetic systems to the full extent. In this context, extending the existing methods is a promising approach.

Although RUP is based on a set of engineering best practices, it is not adapted for ontogenetic software systems. Indeed, RUP deals with changes of requirements only before the software release. Anticipating future changes and building a system to change is not an option in RUP.

In this article, we propose ONTO-RUP as an extension that adapts RUP to the development of the ontogenetic systems and especially Mage-like systems which support a modeling of the changes in the form of a genome that shapes the system in an autonomous way. We are interested, here, in the key phase related to the definition and analysis of the requirements' evolution.

In the remainder of this article, we present briefly, in section 2, limitation of current development methodologies, then, we present the proposed approach ONTO-RUP. Section 5 shows how the extension works with an example. Section 6 compares this work with related ones. Finally, section 7 gives a conclusion and enumerates some research issues.

2 Limitation of Current Development Methodologies

2.1 Examples in Real World

Real world software systems need to evolve continually in order to cope with ever-changing user's requirements. For example, learning software systems have to evolve autonomously and dynamically to deal with evolutions affecting the software after its release, such as:

- Evolution of system's functions, considering anticipated as well as unanticipated ones
- Anticipated evolution consisting of interface evolutions corresponding to several factors (pedagogical, technological, etc.)

- Unanticipated changes for updating some components without stopping the software system.

Another example is the banking transactions management system where evolutions are frequent:

- Anticipated evolution to deal with business rules such as allowing cash withdrawal for some customers even if the account balance becomes negative.
- Unanticipated evolution consisting of new authentication procedures such as using biometric identification devices instead of a credit card.

Ontogenetic systems clearly require suitable development process that deal with evolution as a fundamental concept. A development process for ontogenetic software systems must be provided with mechanisms to:

- **Capture evolution:** A specific process of eliciting anticipated and unanticipated changes that software systems undergo.
- **Modeling evolution:** Describing, structuring, and documenting changes.
- **Deal with implementation:** Including artifacts for coding anticipated changes and providing tools for generating interfaces, assisting unanticipated changes integration within the software code. For example, in the Mage approach, running software consist of a phenotype and a genome. The phenotype corresponds to the traditional code and the genome is composed of elements which continuously and dynamically shape the phenotype. The internal evolutions must be coded in the phenotype using the conventional if-then-else statements which change the system behavior according to some conditions or events.

2.2 Extending the Suitable Method

Methods are organized ways to produce software. They include several steps to follow during the development process, specific representations (graphical or textual), rules governing the system description and design guidelines [18]. In the software engineering domain, there are a number of development methodologies that have been adopted and/or successfully adapted to meet specific business needs. These range from traditional waterfall development to more recent ones like the rational unified process (RUP), and many agile development methods. In our context, rather than proposing a new development process from scratch, we prefer extending existing ones to deal with ontogenetic software development. We choose the RUP process for many reasons:

- RUP is use-case driven which is suitable for us as we propose (later in this article) a modified form of use cases [3, 16] for modeling changes.
- RUP promote the participative development. This is a key feature to deal with cooperative understanding of requirement evolutions.
- RUP is founded by best practices.

2.3 Rational Unified Process

Rational Unified Process is an approach for developing software, which is iterative, architecture-centric, and use-case driven [10]. The RUP is a well-defined and well-structured software engineering process. It clearly defines project milestones, who is responsible for what, how things are done, and when they should be done. The RUP is structured on two axes or dimensions: The dynamic aspect (horizontal) expresses cycles, phases, iterations, and milestones; the static aspect (vertical) expresses activities, disciplines, artifacts, and roles (figure1.).

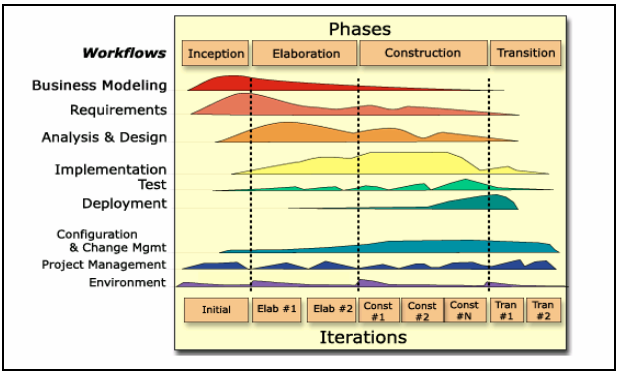

Fig. 1. Dimensions of RUP [10]

2.4 Insufficiencies of the RUP

What is significant in the RUP methodology compared to the others ones is the recognition of the primacy of changes even late in the development cycles compared to the traditional one with the aim of making software development more predictable and more efficient. The ontogenesis implementation requires tools and artifacts to design software system for changing after product release. Unfortunately, RUP and all current development methodologies fall short to deal with ontogenetic software systems.

3 Proposed Solution

A software development process describes *who* is doing *what*, *how*, and *when* [7]; presented in RUP terminology respectively by, workers, activities, artifacts, and workflows. Its adaptation for the ontogenetic systems is relatively a promising and effective approach. We propose an extension called ONTO-RUP which we describe in what follows.

3.1 A New Structure to the RUP

The ONTO-RUP includes, in addition to the usual phases of RUP (i.e. inception, elaboration, construction, and transition), a new phase we have called “Evolution”

that takes place between inception and elaboration phases (Figure 2). The position of the evolution phase in the life cycle corresponds to the level of the progression of the discipline Requirements (Figure1.), we can see that a large portion of Requirements takes place in Inception, although it does continue through to early Transition. In Figure 1, we can observe that at this level, the functional requirements of the system are not all specified (about 20% of the use case model). But, this portion of use case models describes the overall system's behavior which are critical since they have the most important influence on the system's architecture and design.

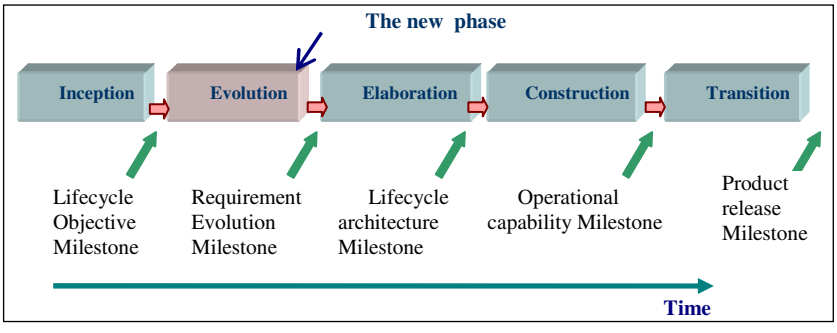

Fig. 2. Phases of the ONTO-RUP

Each phase is concluded with a well-defined milestone; a point in time at which certain critical decisions must be made and therefore key goals must have been achieved. The "Requirement Evolution" milestone supposes that anticipated change cases model is specified in an overall way. During ***Evolution phase*** a large portion of proposed activities in this paper will occur.

3.2 Disciplines for Modeling Ontogenesis

RUP is organized around nine disciplines (Figure 1, vertical axis). A discipline is a collection of activities that are related to a major "area of concern" within the overall project. Disciplines group activities logically. ONTO-RUP proposes a set of disciplines:

- Anticipated evolutions requirement Discipline.
- Evolution Decomposition Discipline.
- Unanticipated evolution Discipline

Our focus, in this paper, is on the "Anticipated evolutions discipline". The purpose of this discipline is to:

- Establish and maintain agreement with the customers and other stakeholders on what the future system should do
- Provide system developers with a better understanding of the system requirements evolution

- Define the boundaries of (delimit) the future system
- Provide a basis for planning the technical contents of iterations.

3.3 Decomposition of Changes

Decomposition of changes deals with anticipated and unanticipated evolutions, and takes into account, for anticipated evolutions, internal ones (built-in the code of the software) and external ones (applied one the software during its execution when some conditions are met). Since anticipated evolutions may be internal or external and this may have an influence on the system performances, it is important to decide which evolution is internal and which one is external. To help developers in achieving this decision, we propose a discipline:"Decomposition Discipline".

4 The "Evolution Phase"

4.1 Objectives

The Evolution phase is the second phase after the inception phase in the life cycle of Onto-RUP. This phase is the backbone of the life cycle, because it deals with the ontogenetic dimension of the system. According to RUP, the inception phase is about understanding the project scope and objectives and getting enough information to confirm that we should/shouldn't proceed with the project. The Evolution phase will take place as the second phase during which there will be a study of the needs for evolution to give the comprehensive view of all aspects related to the evolution of the future system. The Evolution phase milestone presents a well-defined set of objectives. We enumerate the objectives of this phase:

- Understand the overall system evolution
- Eliciting evolution of requirements and elaborating the change case model (a modified form or use case that describes a change)
- Deciding if changes will be internal or external evolutions.

4.2 Inputs/Outputs of Evolution Phase

Figure 3 shows the input and the output of the evolution phase. In this phase change case model is globally elaborated and decisions are made about the nature of the major requirements evolution.

Among the objectives of the RUP inception phase we find the comprehension of the system to be built and the identification of the system fundamental functionalities [9, 10].

The use case model adopted by RUP presents the formal specification of requirements. This model separates the system in actors and use cases.

The Requirement discipline provides the system developers with a better understanding of the system requirements, delimits the boundaries of the system, provides a basis for planning the technical contents of iterations, provides a basis for estimating cost and time to develop the system, etc.

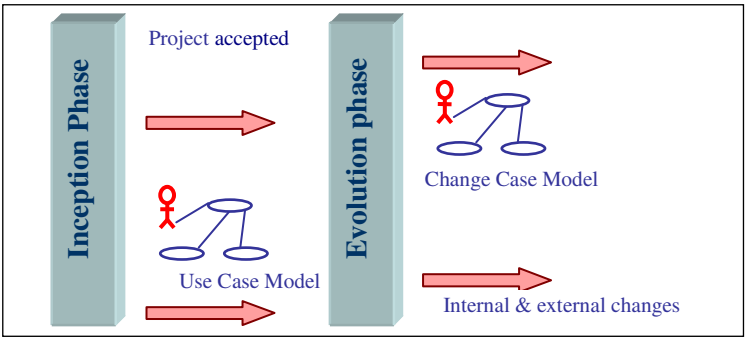

Fig. 3. Inputs/Outputs of the Evolution Phase

Use cases. They are specification of a sequence of actions and variants that a system (or other entity) can perform when interacting with external actors. Each use case describes the behavior of the system under various conditions and shows how it responds to a particular request from an external actor (called the *primary actor*). The primary actor initiates an interaction with the system to accomplish some goal, and the use case is a black box which describes the reaction of the overall system, without indicating how the system achieves the goal.

A complete set of use cases specifies all the possible use of a system and, consequently, describe all the necessary behavior of the system. In our approach**,** we use the use case formalism given by Cockburn in [2].

4.3 Activities of Anticipated Evolutions Requirement Discipline

Starting from use cases model elaborated in the inception phase, the purpose of the "Anticipated evolutions requirement Discipline" is to built the change cases model starting by use case models. These activities require an iterative process, where each performed iteration represents a unit of change which consists of one or more change cases. A unit of change will be an input for the sub-process called *changes study* using techniques for eliciting evolution/change requirements. We find in [13], a set of techniques for eliciting requirements. There are many other techniques and their combination seems to provide richer and more detailed requirements for evolution of requirements. We propose the following combination: begin with *questionnaires*, followed by *structured interview* to gain deeper insight into the possible evolution, then the *brainstorming* technique is used, and at the end the *group work* technique. his ast technique is appropriate in our context, because it suggest collaborative meeting that involve and commit the stakeholders directly and promote cooperation.

Figure 4 shows the steps composing the evolution phase. In the change study step we need to decide if the unit of change is further considered or ignored. In the next step, we extract the change cases model using the change case artifact we have proposed (described next). This formalism is based on the use case one but contains specific parts to deal with evolution.

Change cases. The change case is a powerful formalism to capture potential changes that has been first proposed by Ecklund in [3]. In our work we have introduced specific parts to deal with new requirements but also to specify modification to the already implemented ones. Change cases are characterized by both their simplicity and expressive power. Like the use cases, they concern the system in the whole.

4.4 Planification of the Evolution Phase

\In the evolution phase the first iteration aims at specifying the change cases in a overall way.

Those which follow will accentuate in studying the impact of the specified changes. The Requirements and Ontogenesis disciplines are related to other process disciplines: Analyze and Design, Test, Configuration and Change Management, and Project Management. The use case model, change case model, and Requirements Management Plan are important inputs to the iteration planning activities. Since a use/change case describes how a user will interact with the system, the use of UML notation [16] such as sequence diagrams or collaboration diagrams to show how this interaction will be implemented by the design elements. We can also identify test cases from a use/change case. This ensures that the services the users expect from the system are really provided.

5 Illustrative Example

One of the possible ways to evaluate software development process models or methodologies is to choose some exemplar systems as case studies and employ the process model or methodology in developing case study systems. Let us consider the banking transactions system previously introduced. In the following we describe the two phases of ONTO-RUP.

Inception Phase. In ONTO-RUP the Inception phase is typically equivalent to RUP approach. It provides rich set of activities and guidelines.

> At the end of the inception phase, if there is an agreement about the project (does or not pass this milestone ”Lifecycle Objective”), it can either be cancelled or it can repeat this phase after being redesigned to better meet the criteria. The resulting artifact is a model of use cases which presents an overall schema of current requirements (Figure 5).

Evolution Phase. The analysis of the change of requirements is held during the evolution phase using the change process. After the study of the possible anticipated changes using the techniques proposed in the previous section; decisions are taken concerning the anticipated changes that may affect the software system. Figure 7 summarizes all the use cases composing the global view of the future system. We use

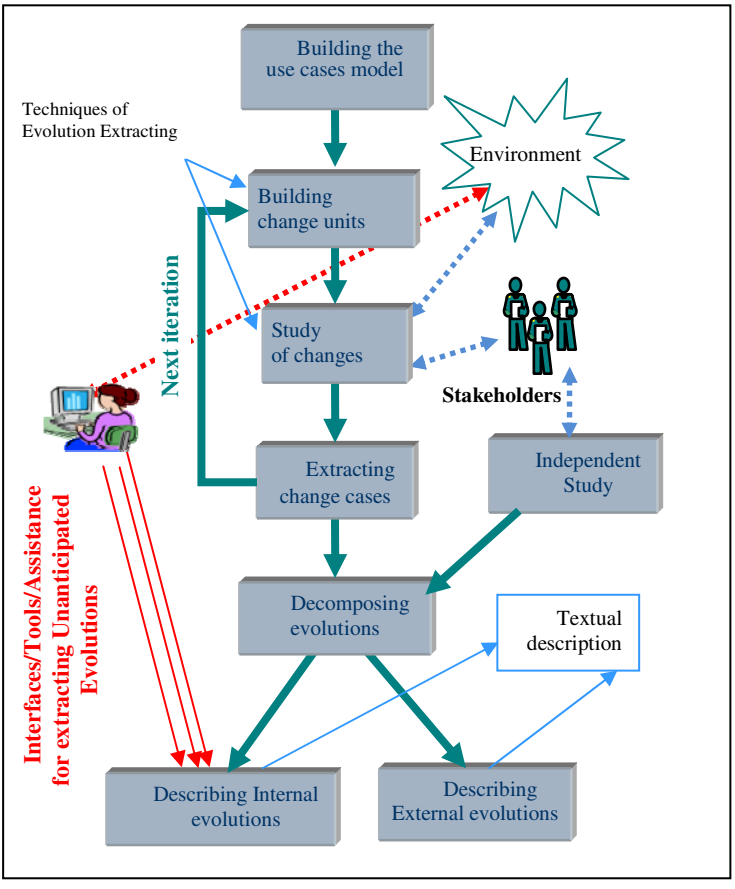

Fig. 4. Requirements evolution Process

the change case formalism suggested in the previous section for describing the anticipated changes. We give two change cases modeled for this system:

- Change case for adding an authentication procedure.

Change case for allowing some customers to make cash withdrawal even with a negative account balance.

- The second change case is an unanticipated change, that's why it doesn't appear in Figure 6. Notice that the change process remains the same for unanticipated changes.

Only few works deal with evolution as an important challenge. Mage considers the evolution as a fundamental process and provides several concepts to capture and model changes as a dynamic and autonomous process [14, 15]. The autonomic computing approach seeks to provide systems with aptitudes to be self-managed and self-repaired [17]. To our best knowledge there is no complete development approaches dedicated to autonomic computing. The same is true for

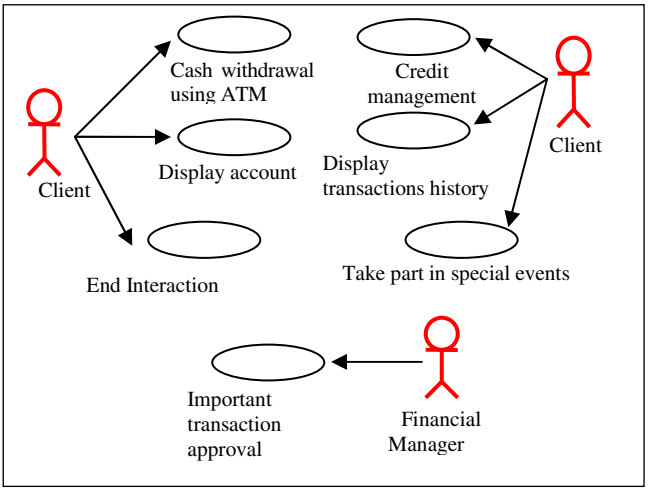

Fig. 5. Overall model of banking transaction system

Use case	Type of change	Description of Change
ATM cash withdrawal	Change	New authentication procedure
Display account	Maintain	No need of change for this use case
Request Credit	Maintain	No need of change for this use case
Credit management	Change	A potential client will benefit of an important credit
Important transaction approval	Maintain	No need of change for this use case
ATM cash withdrawal	Change	Changing the maximal of amount withdrawal of X every 3 years
Fidélisation Order 1	Change	Adding unit of fidelisation If date-date create=1
Fedilisation Order 2	Change	To encourage using ATM benefit> Y
ATM cash withdrawal	Change	Permit supply credit in 15/12 until 31/12 (End year)

Fig. 6. Studied use cases

6 Related Work

Only few works deal with evolution as an important challenge. Mage considers the evolution as a fundamental process and provides several concepts to capture and model changes as a dynamic and autonomous process [14, 15]. The autonomic computing approach seeks to provide systems with aptitudes to be self-managed and self-repaired [17]. To our best knowledge there is no complete development approaches dedicated to autonomic computing. The same is true for biomorphic systems described in [12]. The work in [4], [5] proposes an approach with an evolution

process at the requirement level that uses the concept of gap. This approach is based on a meta-model and a generic typology of operators to express different kinds of evolution.

7 Conclusion

The ontogenetic software systems are particular systems which allow dynamic evolution of requirements. Motivations for research in this context are important and investigations are widely justified from the industrial and economic point of view since at least 50% of software costs are relative to their evolution. However, today, there are no development methods to support such systems.

We presented in this article an attempt in this direction, it proposes an extension of RUP which preserves its features while providing artifacts and methods to support the development of the ontogenetic systems.

As a perspective to this work, we need first to describe other disciplines related to *Modeling Ontogenesis Disciplines*. Another important perspective is to consider the evolutions in subsequent phases of the development process.

References

1. Chapin, N., Hale, J.E., Khan, K.M., Ramil, J.F., Tan, W.G.: Types of software evolution and software maintenance. J. Software Maintenance and Evolution: Research and Practice 13, 3–30 (2001)
2. Cockburn, A.: Writing Effective Use Cases. Addison Wesley, New York (2001)
3. Ecklund, E.F., et al.: Change Cases: Uses Cases that Identify Future Requirements. In: Proceeding OOPSLA 1996. ACM Press, New York (1996)
4. Etien, A., et al.: Overview of Gap-driven Evolution Process. In: AWRE 2004 9th Australian Workshop on Requirements Engineering (2004)
5. Salinesi, C., Etien, A., Wäyrynen, J.: Towards a Systematic Propagation of Evolution Requirements in IS Adaptation Projects. In: Australian Conference on Information Systems (ACIS), Hobart, Australia (December 2004)
6. Jackson, M.A.: System Development. Prentice-Hall, Englewood Cliffs (1983)
7. Jacobson, I., Booch, G., Rumbaugh, J.: The unified software development process. Eyrolles (2000)
8. Johson, B., Woolfolk, W.W., Miller, R.: Flexible Software Design. Auerbach Publications (2005)
9. Kroll, P., Royce, M.: Key Principles for Business-Driven Development. Rational Edge (2005)
10. Kroll, P., Kruchten, P.: The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP. Addison Wesley, Reading (2003)
11. Lehman, M.M.: Laws of software evolution revisited. In: Montangero, C. (ed.) EWSPT 1996. LNCS, vol. 1149, pp. 108–124. Springer, Heidelberg (1996)
12. Lodding, K.N.: Hitchhiker's Guide to Biomorphic Software. QUEUE (June 2004)
13. Loucopoulos, P.: Engineering and Managing Software Requirements. Springer, Heidelberg (2005)

14. Meslati, D.: Mage: Une approche ontogénétique de l'évolution dans les systèmes logiciels critiques et embarqués. Phd thesis, University of Annaba (February 2006)
15. Meslati, D., et al.: The MAGE Ontogenetic Model: Towards autonomously-developed software. In: 17th Int. Conf. on Software & Systems Engineering and their Applications, Paris (November 2004)
16. Muller, P.A., Gaertner, N.: Modélisation objet avec UML. Eyrolles (2000)
17. Murch, R.: Autonomic Computing On Demand Series. Prentice Hall PTR, Englewood Cliffs (2004)
18. Sommerville, I.: Software Engineering. Edition (2000)
19. Rational Unified Process Fundamentals, Instructor Manual Version 2000.02.10, http://cjy.nyist.net