Multi-Agent Spiral Software Engineering : A Lakatosian Approach

Christophe Schinckus[+], Yves Wautelet[*], Manuel Kolp[*]

[+] *CEREC – Centre de Recherche en Economie*
Facultés Universitaires St Louis
Email: {schinckus@fusl.ac.be}

[*] *IAG – Institut d'Administration et de Gestion, ISYS – U*nité de Systèmes d'Information, Université Catholique de Louvain, 1 Place des Doyens, Belgium
Email: {wautelet, kolp@isys.ucl.ac.be}

Abstract. This paper presents an epistemological approach for the development and validation of an original agent oriented software development methodology (see [Wautelet05a, Wautelet05b]). Agent orientation has been widely presented as a new modeling, design and programming paradigm that could be adopted to build systems mark to the determinant advantages it offers. This will be exposed and put into perspective in the paper through the Lakatosian approach. Spiral development (see [Boehm00a]) has become popular, especially through object-oriented software project development since it allows efficient software project management, continuous organizational modeling and requirements acquisition, early implementation, continuous testing and modularity, etc.. The iterative nature of this requirements engineering process will be studied here through Herbert Simon's bounded rationality principle and Popper's knowledge growth principle but nuanced by Lakatos' falsification principle criticism.

1 Introduction

Information systems are deeply linked to human activities. Unfortunately, development methodologies have been traditionally inspired by programming concepts and not by organizational and human ones. This leads to ontological and semantic gaps between the systems and their environments. The adoption of *Multi-Agent Systems* (*MAS*) helps to reduce these gaps by offering modeling tools based on organizational concepts (actors, agents, goals, objectives, responsibilities, social dependencies, etc.) as fundamentals to conceive systems through all the development process. Moreover, software development is becoming increasingly complex. Stakeholders' expectations are growing higher while the development time has to be as short as possible. Project managers, analysts and software developers need adequate processes to model the organizational context, capture requirements and build efficient and flexible software systems. Facing user-interactive software applications this objective will be better achieved using a Spiral *Software Development Life Cycle* (*SDLC*) [Boehm88]. Indeed, the use of such development methodology allows to capture and to refine requirements continuously and consequently to efficiently deal with users' and stakeholders' expectations appearing continuously during the project life cycle.

This paper presents an epistemological approach for the development and validation of an original agent oriented software development methodology (see [Wautelet05a, Wautelet05b]). Agent orientation has been widely presented as a new modeling, design and programming paradigm that could be adopted to build systems mark to the determinant advantages it offers. This will be exposed and put into perspective in the paper through the Lakatosian approach. Spiral development (see [Boehm00a]) has become popular, especially through object-oriented software project development since it allows efficient software project management, continuous organizational modeling and requirements acquisition, early implementation, continuous testing and modularity, etc.. The iterative nature of this requirements engineering process will be studied here through Herbert Simon's bounded rationality principle and Popper's knowledge growth principle but nuanced by Lakatos' falsification principle criticism.

The paper is organized as follows. Section 2 presents the context of the research, some reasons of the software crisis are presented, spiral development and agent-oriented systems as well as their benefits are studied. In section 3, the epistemological approach is exposed. Agent–Orientation is successively considered as a paradigm and as a research program; the iterative nature of the requirements engineering process is also studied. Finally, some conclusions on our approach will be taken.

2 State of the art

2.1 The Software Crisis

Although constant and huge progress has been made in information technology, complex information systems often imperfectly match users' requirements. The "software engineering crisis"-diagnostic made in the late sixties remains up to date. Computers often take the responsibility to the public ("it is the computer's fault!"), but often the true dysfunctional causes are the result of human decisions or methodological insufficiencies.

Software engineering (SE) can be defined as the whole of the methodological processes and of the relevant products used for the development of software on a large scale. It was created to provide a more suitable methodological research.
The crisis has different reasons:

- the **increasing complexity of modern information systems**: continuous technological progresses allow to develop huger and huger software applications while the increasing demand exceeds the productivity gains obtained by methodological, techniques and tools improvements;
- the **common under-estimation of the difficulty and consequently of the software development cost** (the methods, widely empiric, of individual programming are not applicable to the development of huge and complex systems), with consequently a complex, long, costly, conflicting software development process and inadequate software products;
- a **weaker reliability and maturity of software compared to hardware** whereas its relative importance in the realization of the complex system functions is getting higher;
- a substantial delay of the software industry compared to what a good application of the principles would permit (see [Davis95, Meyer97, Sommerville92]). This delay is due to the complexity and to the teaching costs of software developers in a constantly evolving discipline;
- an **inherent complexity**:
  - the problems software has to deal with can be arbitrary complex; for example, the limits of a system functionalities are often much less clear than those of more tangible products;
  - the sequential decomposition of the development phases is not so natural for SE than it is for other engineering disciplines (mechanical engineering for example).

2.2 Software Development Methodologies

A *Software Development Methodology* (SDM) or a *System Development Life Cycle* (SDLC) is a conceptual model used in software project management [Royce98] to describe the stages involved in an information system development project; from an initial feasibility study through maintenance of the completed application. Several development methodologies using various SDLC such as the Sequential/Waterfall Model [Royce70], the V-Model [Forsberg97], the Incremental Model [Dorfman97], the Evolutionary Model or the Spiral Model [Boehm88, Boehm94] were developed over the years. Depending on the project, some SDLC appear to be more adequate than others. For the development of huge and complex user-interactive enterprise information systems the Spiral SDLC is more appropriate than the use of a traditional Waterfall SDLC. Indeed the former is much less risky than the later, especially in the last stages of the project. The aim of this section is to discuss this assumption.

2.2.1. Waterfall SDLC

The Waterfall SDLC [Royce70] describes a development process based on a series of phases (often called disciplines) that are fulfilled one after another in a linear/sequential way. This type of SDLC can only be successful when a series of assumptions are met at the same time. Otherwise, risks are introduced in the process. Following [Boehm00a], these assumptions are:

- the requirements can be known in advance of implementation;
- the requirements have no unresolved high-risk implications, such as risks due to cost, schedule, performance, safety, security, user interfaces, and organizational impacts;
- the nature of the requirements will not change very much either during development or evolution;
- the requirements are compatible with all the key system stakeholders' expectations, including users, customer, developers, maintainers, investors;
- the right architecture for implementing the requirements is well understood;
- there is enough calendar time to proceed sequentially.

If these assumptions are not met, the initial design will likely be flawed with respect to its key requirements and late discovery of design defects results in costly overruns and/or project cancellation. Consequently, time and money will be wasted specifying and implementing requirements that are going to change and implementing a faulty design. This idea is described in the next section.

2.2.2. Spiral SDLC

Following [Boehm00a], the Spiral SDLC is *"a **risk**-driven **process model generator** used to guide multi-stakeholder concurrent engineering of software intensive systems. It has two main distinguishing features. One is a **cyclic** approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk. The other is a set of **anchor point milestones** for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions."*

As argued in the previous section, the assumptions needed for an optimal Waterfall SDLC are seldom met. The causes are:

- **requirements can rarely be known and defined in advance** of implementation, especially for new user-interactive systems. Most of them suffer from what is known as the *IKIWISI* syndrome [Boehm00b]. Users and stakeholders are often, during the early stages of the project, unable to express and describe what their requirements really are. When they are asked about them they often reply *I can't tell you, but I'll know it when I see it* (*IKIWISI*). In such situations, a parallel prototyping – requirements definition – architecture design – implementation – test is fundamental for the good achievement of the project;
- defining requirements and freezing them early on in the project is a very risky task. Early defined requirements can meet users' expectations but sometimes analysts discover later in the project that their practical achievement is (too) constraining (in terms of cost, development time, processing time, human resources, etc.). These requirements expressed in a less constraining manner (i.e. longer response time from the system, less precision in the data analy-

sis, etc.) keep meeting user's requirements but reduce the identified constraints in a drastic manner. That is why it is important **not to fix and freeze requirements too early** in the project but to keep the **ability to refine them** later if their definition appears to be too constraining. Using a Spiral SDLC allows to refine requirements many times during the project lifecycle;

- **requirements** often **evolve** during the development process. Stakeholders express new ideas, needs or perspectives so that the requirements acquisition process cannot be made only once in the early stages but must be a continuous process. This development-method also avoid spending a huge time on requirements analysis at the beginning of the project specifying requirements that will appear to be out of date or be refined later in the project.

These remarks highlight the need for a SDLC that include continuous requirements acquisition and modeling. Using a Spiral SDLC implies that risk is considered during the early stages of the project not at the late ones as in a process driven by fully sequential activities.

2.3 Towards More Adequate Organizational Modeling Techniques: Agent-Orientation

One solution to the SE-crisis can thus be the use of SE methodologies using a spiral SDLC more adequate for the development huge user-centered applications. Another one can be improvements in conceptual modeling. Indeed, the meteoric rise of Internet and World-Wide Web technologies has created overnight new application areas for enterprise software, including eBusiness, web services, ubiquitous computing, knowledge management and peer-to-peer networks. These areas demand software that is robust, can operate within a wide range of environments, and can evolve over time to cope with changing requirements. Moreover, such software has to be highly customizable to meet the needs of a wide range of users, and sufficiently secure to protect personal data and other assets on behalf of its stakeholders.

Not surprisingly, researchers are looking for new software designs that can cope with such requirements. One promising source of ideas for designing such business software is the area of multi-agent systems. Multi-agent system architectures appear to be more flexible, modular and robust than traditional including object-oriented ones. They tend to be open and dynamic in the sense they exist in a changing organizational and operational environment where new components can be added, modified or removed at any time.

Multi-agent systems are based on the concept of **agent** which is defined as "*a computer system, situated in some environment that is capable of flexible autonomous action in order to meet its design objective*" [Wooldridge95]. An agent exhibits the following characteristics:

- **Autonomy**: an agent has its own internal thread of execution, typically oriented to the achievement of a specific task, and it decides for itself what actions it should perform at what time.
- **Situateness**: agents perform their actions in the context of being situated in a particular environment. This environment may be a computational one (e.g., a Web site) or a physical one (e.g., a manufacturing pipeline). The agent can sense and affect some portion of that environment.
- **Flexibility**: in order to accomplish its design objectives in a dynamic and unpredictable environment, the agent may need to act to ensure that its goals are achieved (by realizing alternative plan). This property is enabled by the fact that the agent is autonomous in its problem solving.

Agents can be useful as stand-alone entities that delegate particular tasks on behalf of a user (e.g., personal digital assistants and e-mail filters [Maes94], or goal-driven office delivery mobile devices [Mataric92]). However, in the overwhelming majority of cases, agents exist in an environment that contains other agents. Such environment is a multi-agent system (**MAS**).

In MAS, the global behavior derives from the interaction among the constituent agents: they cooperate, coordinate or negotiate with one another. A multi-agent system (MAS) is then conceived as a society of autonomous, collaborative, and goal-driven software components (agents), much like a social organization. Each role an agent can play has a well defined set of responsibilities (goals) achieved by means of an agent's own abilities, as well as its interaction capabilities.

This *sociality* of MAS is well suited to tackling the complexity of today's organization software systems for a number of reasons:

- it permits a better match between system architectures and their operational environment (e.g. a public organization, a corporation, a non-profit association, a local community, …);
- the autonomy of an agent (i.e., the ability an agent has to decide what actions it should take at what time [Wooldridge95]) reflects the social and decentralized nature of modern enterprise systems [Tennenhouse00] that are operated by different stakeholders [Parunak97];
- the flexible way in which agents operate to accomplish their goals is suited to the dynamic and unpredictable situations in which business software is now expected to run [Zambonelli00a, Zambonelli00b].

3 Epistemological Approach

In the second part of this paper, we will show that a lakatosian methodology is well adapted for an epistemological reading of the emergence, on the one hand, of agent-orientation and, on the other hand, of spiral development.

We will firstly reconsider the literature use of the “paradigm”-concept to characterize the evolution from object-orientation to agent-orientation. We will explain why we think that the lakatosian “research program”-concept is more adequate than the kuhnian “paradigm”-concept to illustrate the evolution of modeling and programming concepts.

We will also reject the use of the popperian model of knowledge, which is often used to explain the iterative process of computer development. According to us, this popperian vision of computer development is inadequate because the “quasi-empirical” character of software development does not allow any radical falsification of computer solutions. We will show that the lakatosian epistemology provides a very interesting and accurate theoretical framework to analyze the iterative process inherent to spiral development.

3.1 Agent-Orientation: “Paradigm” or “Research Program”?

The evolution from object-orientation to agent-orientation is often presented in the literature as an evolution from one paradigm to another. In this section, we will reconsider the definition of the “paradigm”-concept in order to show that its use to characterize the evolution of computer science is often abusive. Afterwards, we will demonstrate that the use of the lakatosian “research program”-concept is more adequate for an epistemological analysis of an applied discipline such as computer science.

3.1.1 Kuhnian Vision of Computer Science

3.1.1.1 “Paradigm”-Concept: a Definition

The word “paradigm” comes directly from the philosophical world where its meaning remains surprisingly rather vague. As reminded by Göktürk and Akkok [Göktürk04], Plato and Aristote were the first authors to introduce this concept. According to them, the paradigm is a kind of explanatory model, which allows people to understand, in terms of causality, the changes imposed by Nature. However, the paradigm is not, strictly speaking, a logic. For Aristote, the “paradigm” was “different from both deduction, which goes from universal to particular, and induction, which goes from particular to universal, in the sense that the paradigm goes from particular to particular.” [Göktürk04].

The term “paradigm” has not really been used before the 20th century when Thomas Kuhn developed a specific epistemology based on this concept. According to Kuhn, the paradigm is defined as “*a constellation of concepts, values, perceptions and practices shared by a community and which forms a particular vision of reality that is the basis of the way a community organizes itself* ” [Kuhn96]. Nevertheless, Kuhn himself admits that the use of the word remains rather vague. According to Masterman [Masterman70], it is possible to identify twenty-two different meanings of the “paradigm”-concept used in the kuhnian epistemology. In the last edition of his book, Kuhn even recognized that the “paradigm”-concept is vague but he explained that it is close to what he calls a “disciplinary matrix”.

That is why we will consider in this paper the “paradigm” as *a way of representing the world, which necessarily includes conceptual tools and methods (the conjunction of these two elements forming what Kuhn called a disciplinary matrix), such that an observer can create models*. Each paradigm refers to a particular ontology and represents a subset of “what is representable”. The representation abilities of a paradigm are basically related to the conceptual tools, to the modelization methodology and to the use of these two elements by theoreticians.

3.1.1.2 Paradigms and Computer Science

Göktürk retraces briefly the evolution of the paradigms in computer science and explains that the first one in computer programming was the **procedural paradigm** [Göktürk04]. This paradigm was based on the use of algorithms to execute particular tasks. The second paradigm was the **data-hiding paradigm**, which was focused on the data’s organization and introduced the concept of modules (to hide the data’s). This paradigm was followed by the **data-abstraction paradigm,** which was concentrated on the types and on the operations defined on these types. Next paradigm was the **object-oriented paradigm**, “built upon the data-abstraction” one [Göktürk04] but introducing new concepts like heritage or polymorphism. Finally, using the flexibility of the component-oriented logic, the **agent-oriented paradigm** has divided software into independent and communicating entities called “agents” [Woolridge95]. This last paradigm has been described in detail in the first part of this paper.

Programming languages and modelization paradigms are interdependant. We could use the “chicken and egg” metaphor to characterize their reciprocal relationship [Göktürk04] : sometimes, specific needs for a programming language leads to a better implementation of a modelization paradigm and sometimes, the evolution of the modelization paradigm influences and improves the development of a specific programming language. However, even if the programming languages and the modelization paradigms are interdependent, an important specificity of the agent-oriented paradigm is that it is not formally related to its own programming language. The concepts used in agent-orientation have been inspired from the organizational structures of the real world. At the beginning, the agent-oriented models were implemented in object-oriented languages but further evolutions allow to implement the multi-agents systems directly in terms of *Beliefs*, *Desires* and *Intentions* (*BDI*) (see [Jack, Jadex]). However, these languages are nowadays still based on object-oriented architectures.

3.1.2 Lakatosian Vision of Computer Science

In the literature, the paradigm-concept is often used (for example in [Göktürk04] and in [Castro02]) to explain, in methodological terms, the emergence of agent-orientation. In the following sections, we propose to review this vision that is to demonstrate that the paradigm-concept is not the best adapted to describe the emergence of agent-orientation. After presenting the concept of research program developed by Lakatos, we will explain why, according to us, a lakatosian reading of the evolution to agent-orientation is more appropriate than a kuhnian one.

3.1.2.1 "Research Program"-Concept: a Definition

In the continuity of the popperian philosophy (which will be briefly presented in the following section), Imre Lakatos has developed in 1974 an original approach of science. He considers scientific theories as general structures that he calls "research programs". A lakatosian research program is a kind of scientific construction, a theoretical framework which guides future research (in a specific field) in a positive and negative way. Each research program is constituted by an *hard core*, a *protective belt of auxiliary hypotheses*, a *positive heuristic* and a *negative heuristic*.

The **hard core** is composed by general theoretical assumptions which constitute the basic knowledge for the program development. In other words, these axioms are the assumptions that the theorists will not challenge in their later researches. This hard core is surrounded with a **protective belt** composed of the auxiliary hypotheses, which complete the hard core and with assumptions related to the description of the initial conditions of a specific problem. These auxiliary hypotheses will be thoroughly studied again, widened and completed by theorists in their further studies within the program. This widening of the protective belt hypotheses contributes to the evolution of the research program without calling into question the basic knowledge shared by a scientific community.

The **positive heuristic** represents the agreement among the theoreticians over the scientific evolution of the research program. It is a kind of "problem solving machinery" composed by proposals and indications on the way to widen and to enrich the research program. The **negative heuristic** is the opposite of the positive heuristic. Within each research program, it is important to maintain the basic assumptions unchanged. It means that all the questions or methodologies that are not in accordance with the basic knowledge must be rejected. All doubts appearing about the basic knowledge of the main theoretical framework become a kind of negative heuristic of the research program. When the negative heuristic becomes more and more important, a research program can become "degenerative" (i.e. it has more and more empirical anomalies). This means that theoreticians have to reconsider the basic knowledge of the program, which can lead to the creation of another research program. Let us mention that this revision is always a very slow process.

According to Lakatos, we can characterize the evolution of knowledge as a series of "problems shifts" which allow the scientific theories to evolve without rejecting the basic axioms shared by theorists within a specific research program. The concept of "research program" represents a descriptive and minimal unit of knowledge, which allows for a rational reconstruction of the history of science.

At first sight, the "research program"-concept seems rather close to the "paradigm"-concept. Indeed, it is, in both cases, a matter of "disciplinary matrix" used to describe a particular ontology of the external world. However, differences exist between these two concepts especially in the evolution of science and knowledge in a large sense.

According to Kuhn, the evolution of science does not follow a straight line and does not converge towards something, which would be the « truth ». In the kuhnian vision, the evolution of science could be represented by a broken line where discontinuity would mark the passage from one paradigm to another. From this point of view, different paradigms cannot be compared. Moreover, Kuhn specifies that a paradigm always emerges within a discipline facing a methodological crisis (characterized by the absence of a dominating theoretical framework) [Kuhn96]. Following a crisis undergone by a previously dominating paradigm, a new paradigm emerges with a new language and a new rationality. This new way of thinking the world does not allow a comparison between the old and the new paradigm. Given that a new paradigm is a new way of thinking the world, there is no comparison base. The thesis of paradigms incommensurability has become a very well known issue in the philosophy of sciences [Sankey94].

Lakatos decomposes the evolution of science into successive methodological and epistemological steps. These steps form a kind of vertical structure built with a multitude of "layers of knowledge" and where each layer represents a particular research program. In the lakatosian vision, the emergence of a new research program is induced by an empirical degeneration of a previously dominating research program. The new research program will constitute a superior layer of knowledge, which will integrate the same conceptual tools as the former one but which would be able to solve its empirical anomalies through what Lakatos calls a "problems shift". "Problems shifts" are characterized by an extension or a redefinition of the protective belt of the preceding program. In this vision, research programs remain comparable to each other (in both conceptual and empirical terms). The language and rationality of the new research program result from a progressive evolution of knowledge and from the resolution of the empirical anomalies of the previous research program. In opposition to the kuhnian vision, Lakatos explains that there is no discontinuity between the different research programs.

3.1.2.2 Agent-Orientation as a Research Program

Although some authors present the emergence of the agent-orientation as an evolution towards a new paradigm, we propose to use the lakatosian concept of research program to characterize this evolution. In this section, we present three main arguments for the use of the lakatosian research program to understand the shift from object to agent-orientation.

- Kuhnian Crisis or Lakatosian Problem Shift?

As exposed in the first part of this paper, we observe a SE-crisis due to the fact that little software projects really manage to satisfy users' requirements. In the kuhnian vision, this crisis could be considered as a favourable argument for the emergence of a new paradigm. In this perspective, an interesting question is: *does the current crisis characterize the end of a dominating paradigm or is it simply the result of a pre-paradigmatic step specific to "young sciences" which have not found a dominating paradigm yet?* Using the kuhnian rhetoric to analyse this crisis, we can consider the situation as a paradigm evolution. Indeed, the pre-paradigmatic step in computer science was rather characterized by the procedural framework (which was defined by an algorithmic and sequential logic i.e. a strictly computer/mathematical logic) as well as the data-hiding and the data-abstraction frameworks. This pre-paradigmatic period was essential for the evolution towards object-orientation. However, the crisis situation observed in SE must be carefully analysed. Even if the "SE-crisis" diagnostic has been noted for several years, we think that the context in which agent-orientation has emerged cannot be considered as a crisis in the kuhnian sense. Indeed, most methodological rules existed before agent-orientation and the current software development field does not seem to be so chaotic: IT specialists dispose of analysis methods and methodological tools with a high level of abstraction. These elements tend to show that what looks like a crisis is rather an (animated but normal) evolution of knowledge in computer science, which could be interpreted as a "problem shift" in the lakatosian vision.

- Kuhnian Discontinuity or Lakatosian Continuity?

According to us, the kuhnian discontinuity between paradigms is not appropriate to explain the emergence of agent-orientation because there is no real "fracture" between object and agent-orientation. Indeed, in some computer solutions, communicating objects are used and are completely relevant and sufficient. If we would like to use agents in these solutions, the agents would behave exactly as objects: they would just transfer messages and would not behave like learning and collaborating agents pursuing goals. In this special case, there is no contribution of the agent concept to the computer solution provided by the previous technology based on the object concept. We can see that the cohabitation within the same application between modules exploiting object technology and modules exploiting agent technology is thus a solution which can be "optimal". In a lakatosian vision, this cohabitation represents a progressive evolution of knowledge in computer science. Indeed, the lakatosian epistemology implies that the transition between research programs is not clear and depends on the specific aspects of the experiment made (computer solution in the software field). An hybrid solution between modules developed on the basis of objects and modules developed on the basis of agents can thus be explained by the continuity between research programs inherent to the lakatosian vision of knowledge applied to SE.

- Kuhnian Incommensurability or Lakatosian Commensurability ?

Another drawback to the use of the kuhnian epistemology in SE is the incommensurability between paradigms. Object-orientation and agent-orientation can be compared since collaborating agents can be used as communicating objects and, more important, agents can be implemented using object-oriented languages. In this perspective agents can be considered as "super objects" i.e. objects possessing skills as collaboration, learning, etc. If we consider SE development as an history of "raising the level of abstraction", agent-orientation can be seen as an evolution of object-orientation because it raises the level of abstraction a little higher. In this perspective research programs preceding object-orientation can also be considered as lower layered than the later. This vision matches perfectly with the lakatosian concept of layers of knowledge introduced earlier. Indeed, considering the SE evolution, each new research program raises the level of abstraction and constitutes a higher layer of knowledge. These layers are comparable so that OO and AO are commensurable.

- Lakatosian Vision

Agent-orientation is based on basic knowledge that existed before its emergence. We could say that the agent-oriented programming has a **hard core** composed of the concepts defined in the previous research programs (procedural, data hiding and data abstraction) on the one hand and by the artificial intelligence field [Wooldridge95] on the other hand. The **protective belt** of the agent-orientation research program would be characterized by the evolution towards a methodology of SE allowing to develop very important projects implying an intensive implication of the users (see for example [Wautelet05, Do03, Faulkner05]).

Table 1 presents a summary of the opposition between the kuhnian and the lakatosian epistemologies applied to the emergence of agent-orientation.

	The kuhnian epistemology : the paradigm	**Critics of the kuhnian epistemology**	**The lakatosian epistemology : the research program**
Emergence of Knowledge : Crisis or "Problem shift" ?	Existence of a methodological crisis allowing the emergence of a new paradigm.	The context in which agent-orientation has emerged cannot be considered as a kuhnian crisis because of the existence of methodological rules preceding the emergence of agent-orientation.	Empirical degeneration of the dominating research program allowing the progressive emergence of a new research program (i.e. "problems shift").
Evolution of Knowledge : Discontinuity or Continuity ?	Emergence of a new paradigm implying a complete redefinition of the concepts and methodologies (rationality) of the former scientific framework.	Agent-orientation seems to be based on the evolution of the previous methodological rules. Continuity between object and agent-orientation is thus existing.	Unchanged hard core based on the previous research programs and on the concepts defined by artificial intelligence. Change in the protective belt showing a methodology evolution towards a higher level of abstraction (new layer of knowledge).

Evolution of Knowledge : Incommensurability or Commensurability ?	Incommensurability between paradigms as a consequence of the knowledge discontinuity.	Object-orientation and agent-orientation are directly commensurable since we can interpret some object-oriented solutions in terms of agent-oriented systems.	Commensurability between two research programs due to the unchanged hard core.

Table 1. kuhnian and lakatosian visions to the emergence of the agent-orientation.

In the light of the arguments presented above, we conclude that the kuhnian epistemology, often refeered to in the literature, is not appropriate to provide a correct epistemological analysis of knowledge in SE. We propose to rather use the lakatosian epistemology to characterize the emergence of the agent-orientation.

According to us, the lakatosian epistemology is directly in line with the idea of computer knowledge depicted as a « structure in layers ». We have represented this architecture in the following Figure 1.

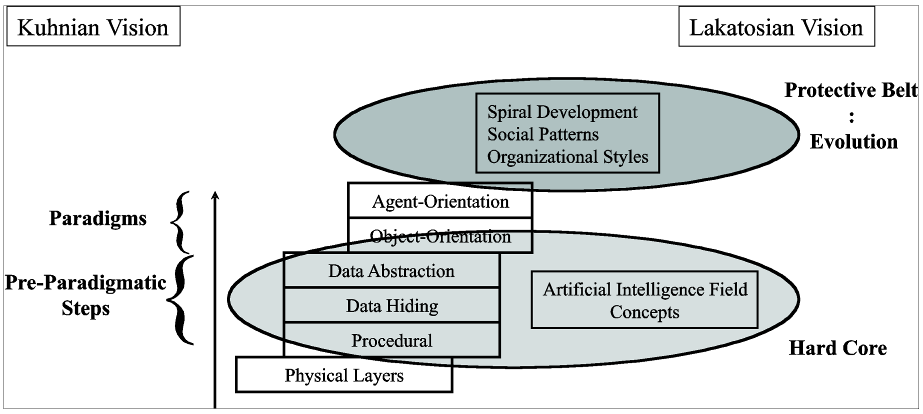

Figure 1. Evolution of Knowledge in SE

3.2 The Iterative Nature of the User Requirements Collecting Process

Adaptive rationality that can be found in the lakatosian epistemology is very useful for a methodological reading of Spiral development. Indeed it is in accordance with the iterative nature of the requirements elicitation process.

According to [Toffolon99], the iterative nature of the requirements elicitation process is based on Herbert Simon *bounded rationality* principle and on Karl Popper *knowledge theory*. In this section, we will show that the principle of bounded rationality is directly in line with the adaptive rationality described in lakatosian epistemology. Inspired by Lakatos, we will reject the use of the popperian theory of the knowledge in applied to Spiral SE.

3.2.1 Bounded Rationality

Herbert Simon (Nobel Prize in economics, 1978) proposes the concept of *bounded rationality* [Simon83] to characterize the human reasoning and behavior in uncertain situations. This concept illustrates the idea that human being has limited abilities to analyze all the parameters of the solutions he/she faces. According to Simon, the rationality of human being is related to the present and to the psychological biases. The rationality suggested by Simon is much broader than the perfect rationality which would be able to evaluate all the present and future parameters of a particular solution. This idea of perfect rationality is often used in economics that is why Simon won a Nobel prize in this field.

The bounded rationality principle is more in accordance with the rationality used in social sciences and this concept has allowed the emergence of behavioral economics which tend to be more and more dominant[1]. In a "bounded rationality" vision, we have to take into account the fact that human being cannot evaluate perfectly the moving contexts and that this mis-interpretation of the situation leads human being to seek a satisfactory solution. This satisfactory solution depends directly on each individual and this seeking of a satisfactory solution allows people to meet their fundamental needs and to develop themselves. In this human reasoning model, no way to determine if this "satisfactory solution" maximizes a specific utility function or to know if another choice would have provided an higher level of satisfaction. This vision of rationality can be used to characterize an organization behavior: given the uncertainty and the complexity of the real world (and the technological progress), there is no universal and optimal software solution (which would be rationality superior to the other possible solutions). Because software engineering is a human practice, the perfect software solution driven by a perfect rationality does not exist. We have to think the computer solution in terms of satisfactory solution which can be improved by an iterative process.

According to Brian Arthur [Arthur94], the concept of perfect rationality is extremely useful to solve some theoretical problems. Perfect rationality presupposes that individuals have much higher computational abilities than they have in the real world. Indeed, beyond a certain level of complexity, the logical abilities of people are not sufficient to evaluate the situation. Moreover, in complex situations implying interactions between people, an individual, who would have perfect rationality, would have to guess the behavior of other people (led by bounded rationality). So human behavior

[1] Kahneman (psychologist of economics) won the Nobel Prize in Economics in 2002.

is always determined by bounded rationality and this rationality does not depend only on the ability to evaluate a situation but also on the interactions with other (non-perfectly rational) people. Let us remind that human relations are partly driven by subjective beliefs which complicate the rational evaluation of a particular situation. We could say the complex interactive situations can be seen as undefined problems. To face such problems, Arthur [Arthur94], in accordance with the behavioral economics (and with the bounded rationality principle), proposes to think the human mind as a simplification machine of the problems. This simplification is possible thanks to internal patterns developed by people. Each actor has a collection of internal beliefs which materialize in a particular evolving behavior. Indeed, actors improve their behaviors and learn which of their internal belief is adapted to face a complicated and undefined situation. We can then observe a kind of iterative improvement of human behavior. According to Kaplan [Kaplan82], the principle of bounded rationality results from a particular *conservative incrementalism* materialized in a careful attitude always concerned by feedbacks provided by human interactions.

The principle of bounded rationality is an interesting concept for the theoretical analysis of software development. Indeed, thanks to the bounded rationality principle we can characterize the complexity observed in software solutions supposed to give an efficient response to organizational problems which are not always well defined. In this vision, the organizational interactions allow users to improve their knowledge in an iterative manner and to improve their requirements comprehension. Spiral software development uses an iterative requirements collecting process to progressively improve the software solution during the project lifecycle.

According to us, the use of the bounded rationality principle in software development enhances a the lakatosian reading of it. Our two major arguments are:

- Bounded rationality illustrates the adaptive character of the human mind and the iterative process of knowledge which results from this adaptive logic. This iterative vision of knowledge is directly in line with the lakatosian idea of a progressive “problems shift” presented in the previous sections;
- In his works about the mathematical formalisms, Lakatos proposes two categories of theories: Euclidean theories and quasi-empirical theories. Whereas the first are connected with an axiomatic approach (imposing a purely deductive logic from a priori axioms), the seconds are determined by the empiricism because they evolve through a continuous feed-back between the theory and its empirical results. As we will show in the next section, this point of view is different from the popperian one because in its epistemology, the theoreticians have to change the whole theory if it is falsified. Once again, we think that the lakatosian epistemology can be useful in SE because his iterative process of theories adjustment is directly in line with the quasi-empirical logic observed in software development.

3.2.2 Knowledge Growth Theory

According to Popper [Popper72], scientific knowledge evolves thanks to a continuous and infinite problems resolution process. Knowledge increases by confronting theories with reality. This confrontation, called *falsification* by Popper himself, often generates new problems that theoreticians have to define, to theorize and to solve. The empirical refutation of the proposed theory provides new problems and so on. The knowledge growth process is ad infinitum because, according to Popper, the resolution of new problems requires new knowledge (i.e. new theories and new conceptual tools). Popper calls "crucial experiment" [Popper59] any theory refutation allowing the emergence of new problems. This refutation constitutes a real source of inspiration for the development of new theories supposed to solve the new problems.

According to [Toffolon99], the popperian knowledge growth model can be applied to iterative software development. Indeed, by disturbing the established relations in a particular organizational context, each software solution generates new problems which are specific to this organization. It is impossible to predict exactly how a new software solution will disturb the organizational context. This context is often very complex because a lot of interactions between its components exists and moreover, an iterative process leads to a moving evolution of this context. At first time, the stakeholders, by using a software solution, modify their requirements to solve the new problems created by this system and then improve their use of the system. Stakeholders requirements cannot be fully described and specified before the implementation because they evolve in an iterative way during the use of the software system. [Toffolon99] explains that this iterative evolution can be characterized by a popperian approach. In the following section, the use of the popperian knowledge growth theory applied to iterative software development will be reconsidered.

3.2.3 Falsification Principle: a Critical View

The aim of this section is to moderate the use of popperian epistemology in SE. At first sight, the popperian knowledge growth model is very close to what Lakatos calls the "quasi-empirical theories". However, the two visions are different and, according to us, only the lakatosian one is adequate for an epistemological analysis of software development. We present our arguments in this section.

Popper developed his epistemology for "hard sciences", especially for physics. That is why he thinks science mainly in terms of "representation of the world" rather than in terms of "efficient answers" as it would be in the field of applied sciences. This detail is from preliminary importance because, at the opposite of physical sciences, an universal law true for every software development cannot exist. There is no universal and optimal computer solution (cfr. principle of bounded rationality): no computer solution is identical to another because each organization is different. The practical software use supposed "to falsify" the developed solution appears to be extremely complex and multiple. In this context, it is impossible to dispose of what Popper calls a "crucial experiment", therefore, we cannot reject all the theoretical bases (like object or agent-orientation, the used requirements elicitation process, etc) of the developed solution in case of failure. Consequently the falsification principle specific to the popperian epistemology cannot be applied in the strict sense to software engineering as it would be to "hard science". In software development, each new solution is a new experiment based on various theoretical concepts and trying to satisfy various requirements. Each computer solution is developed for a particular organization employing various people pursuing various goals and it is very difficult to compare the reasons of an empirical failure. Moreover, as [Priestley2005] explains, it is, in software development, very hard to identify correctly the nature of the error. The reason of an empirical failure can be conceptual or result from a bad representation of the organizational logic but it could simply be the consequence of a programming mistake or to a technical problem in the physical layer. The variety of errors strongly complicates the idea of a "crucial experiment". As we mentioned earlyer, software development is not a pure empirical science but rather a "quasi empirical" applied discipline.

For sure, the empirical failure of a software solution leads to and has to lead to a modification of the theoretical foundations but all the modeling techniques on which the solution is based cannot be directly and purely rejected. The popperian model of knowledge (also called "dogmatic falsificationnism" in philosophy of science) appears to be too radical for an epistemological reading of SE because, in case of failure, it simply suggests to reject the theoretical tools used during that development. This popperian radicality is recognized and caused a plentiful literature in philosophy of science (see [Kuhn96] or [Lakatos77]). In line with the opposition to the radicalism of falsification, Lakatos developed its methodology of "research programs".

The lakatosian epistemology (often "called methodological falsificationnism") is less radical than the popperian one. Indeed, Lakatos tries to explain why scientific theories continue to exist in spite of the existence of empirical refutations. Even if empiricism plays a very important role in the evolution of the knowledge, Lakatos recognizes that all the theories born, are refuted and die refuted. The Hungarian philosopher strongly criticizes the concept of "crucial experiment" and argues it is impossible to explain the evolution of science only in popperian terms. Lakatos recognizes the importance of empiricism since it is precisely through the confrontation to the real world that sciences evolve, however, the philosopher explains why the "ad hoc"[2] characteristic often observed in empirical studies can be useful to improve the theory. In the lakatosian vision of science, it is not necessary to change the whole theory in case of refutation. The empirical failure just represents a sign of a problem which must be solved by improving the tested theory. However, as long as the tested theory does not face a continuous empirical failure (which would mark the degenerative character of the research program in which the theory has been developed), nothing obliges the theo-

[2] Ad hoc is a latin expression which means "*for this purpose*". In philosophy and science ad hoc often means the addition of corollary hypotheses to a scientific theory to prevent this theory from being falsified by anomalies not anticipated by its inital theory. Popper rejects this « ad hoc » character in science. According to [Popper59], this character is a kind of intellectual dishonesty.

rists to modify their theoretical framework[3]. This vision is particularly well adapted to the epistemological study of software development lifecycles.

In this paper, we showed that a lakatosian epistemology would be better adapted than a popperian one to characterize the knowledge evolution in SE. Indeed, the lakatosian epistemology, by rejecting the concept of "crucial experiment" is less radical with the importance of the empiricism and allows a feed-back process between the empirical and the theoretical levels. Moreover, as we mentioned in the previous section, the "quasi-empirical" character of software development and the complexity of organizational entities require flexibility in the interpretation of problems encountered during the implementation of a software solution. Finally iterative development can be explained in epistemological terms by two principles. On the one hand, bounded rationality explains why stakeholders' requirements cannot be determined once for all at the early stages of the project. On the other hand, the lakatosian philosophy explains how software development allows a periodic re-examination of the theoretical tools determined through an adaptive knowledge (due to organizational contexts shifts).

4 Conclusion

Users requirements poorly taken into account as well as modeling and programming languages inspired by programming concepts in spite of by organization and enterprise ones has led to a software crisis. Solutions can be found on different domains, among those domains agent-orientation and spiral development constitutes two promising areas. Agent-orientation furnishes concepts to model the organization more precisely and spiral development allows a more efficient requirements elicitation process.

This paper has presented an epistemological analysis of these two improvements. Often described as "the" new paradigm, agent-orientation is a considerable innovation in tools allowing analysts to model the world. We showed in this paper that the emergence of this innovation can be better explained using lakatosian ideas rather than kuhnian ones. In this vision, agent-orientation would be considered as a research program rather than a paradigm. Even if it seems to be just a matter of terminology, the differences between these two epistemological concepts is clear and profound so that they should not be confused. We confronted the kuhnian and the lakatosian vision of the emergence of agent-orientation in order to show that, contrarily to what is presented in literature, agent-orientation is not a paradigm but a research program.

[3] [Lakatos77] reminds that science is a human practice and that it is sometimes very difficult for the scientists do to reject all their theoretical framework when they have devoted several years of their life to this theoretical framework.

Spiral development is a well-known practice in SE, however, its use remains too low compared to the benefits brought by a more accurate requirements engineering process. This process is generally characterized in literature by the bounded rationality principle and the knowledge growth theory. In this paper we tried to show that a lakatosian analysis of spiral development would be more adequate than a popperian one. Indeed, the theory of the knowledge developed by Popper cannot be applied in software development because, in case of empirical failure ("crucial experiment"), the well-known popperian falsification principle leads to a rejection of the theoretical tools on which the falsified solution is based. The "quasi-empirical" character (defined by Lakatos) of software development combined with the complexity of the organizational context implies the impossibility of the existence of a popperian "experiment crucial", so that, the concept of falsification cannot be applied in software development.

In response to the popperian shortcomings, we propose to use a lakatosian approach to provide an epistemological analysis of, on the one hand the emergence of agent-orientation and, on the other hand, the iterative nature of the requirements engineering process. As regards the emergence of agent-orientation, lakatosian epistemology enables to explain the continuity and the commensurability between this new research program and the older modeling and programming techniques. Furthermore the "quasi-empirical" character developed by Lakatos enables to better include and understand the adaptive and iterative nature of the requirements elicitation process (in conformity with the principle of bounded rationality developed by Simon) on which spiral software development is based.

References

[Arthur94] Arthur W.B., “Inductive Reasoning and Bounded Rationality”, American Economic Association Annual Meetings, 1994.

[Boehm00a] B. Boehm edited by Wilfred J. Hansen, “Spiral Development : Experience, Principles, and Refinements”, Spiral Development Workshop February 9, 2000, July 2000.

[Boehm00b] Barry Boehm, “Requirements that Handle IKIWISI, COTS, and Rapid Change” Computer, IEEE, July 2000, pp. 99-102.

[Castro02] J. Castro, M. Kolp and J. Mylopoulos. “Towards A Requirements-Driven Development Methodology : The Tropos Project.” In Information Systems, Elsevier, 2002.

[Davis95] Davis. A. “201 principles of software development.”, McGraw-Hill, 1995.

[Do03] T. T. Do, M. Kolp and A. Pirotte. “Social Patterns for Designing Multi-Agent Systems”, In *Proceedings of the 15th International Conference on Software Engineering and Knowledge Engineering (SEKE 2003)*, San Francisco, USA, July 2003.

[Dsouza98] D. D’Souza and A. Wills, “Objects, Components, and Frameworks with UML: The Catalysis Approach.”, Addison-Wesley, 1998.

[Faulkner05] S. Faulkner, M. Kolp and P-Y. Schobbens, “An Architectural Description Language for BDI Multi-Agent Systems”, submitted to International Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS), Kluwer, 2005.

[Foucault75] M. Foucault, “Surveiller et Punir: Naissance de la Prison.” Paris: Gallimard, 1975.

[Göktürk04] Göktürk E., Akkok N. “Paradigm and Software Engineering”, In Proceedings of Impact of Software Process on Quality, May 2004.

[Gruber93] T. R. Gruber, “A translation approach to portable ontology specifications.” Knowledge Acquisition, vol. 5, pp. 199-220, 1993.

[Jack] JACK Intelligent Agents, http://www.agent-software.com/.

[Jadex] JADEX BDI Agent System, http://vsis-www.informatik.uni-hamburg.de/projects/jadex/.

[Jennings] Jennings N.R and Woolridge M., "Agent-Oriented Software Engineering," in Handbook of Agent Technology, J. Bradshaw, Ed.: AAAI/MIT Press, (to appear).

[Jureta05] I. Jureta, M. Kolp, S. Faulkner, Y. Wautelet, “A Goal-Oriented Framework for Business Modelization”, Working Paper IAG 126/04, Université Catholique de Louvain, January 2005.

[Kaplan82] Kaplan S., Kaplan R. : “Cognition and Environment : Functioning in an Uncertain World”, Praeger, 1982.

[Kolp97] Kolp M., Pirotte A., Zimanyi E., “Modélisation conceptuelle et ingénierie des systèmes d'information”, In Actes des Journées Francophones Sciences de la Cognition vers les Applications, Villeneuve d'Ascq, France, July 1997.

[Kuhn96] Kuhn T., “The structure of Scientific Revolutions”, 3rd ed: University of Chicago Press, 1996.

[Lakatos74] Lakatos I., “Methodology of Scientific Research Programmes” In Criticism and the Growth of Knowledge, edited by I. Lakatos and A. Musgrave. Cambridge: Cambridge University Press, 1974.

[Lakatos77] Lakatos I., “The Methodology of Scientific Research Programmes: Philosophical Papers Volume 1”. Cambridge: Cambridge University Press, 1977.

[Maes94] P. Maes. “Agents that reduce work and information overload”. Communications of the ACM, 37(7): 30-40, 1994.

[Masterman70] Masterman M., “The Nature of Paradigm”in I. Lakatos et A.Musgrave, Criticism and the Growth of Knowledge, Cambridge, Cambridge University Press, 1970.

[Mataric92] M. Mataric. “Integration of representation into goal-driven behaviour-based robots”. *IEEE Transactions on Robotics and Automatation*, 8:59-69, 1992.

[Meyer97] B. Meyer. “Object-oriented Software Construction.” Prentice Hall, second edition, 1997.

[Parunak97] V. Parunak. “Go to the ant: Engineering principles from natural agent systems”, Annals of Operations Research, 75: 69-101, 1997.

[Popper59] Popper K.R. The Logic of Scientific Discovery. (translation of Logik der Forschung). Hutchinson, London, 1959.

[Popper72] Popper K.R. : “Objective Knowledge, An Evolutionary Approach”,Oxford University Press, 1972.

[Priestley2005] Priestley M. : “The Logic of Correctness in Software Engineering”- Cavendish School of Computer Science, University of Westminster, London - http://users.wmin.ac.uk/priest

[Pylyshyn87] Z. Pylyshyn. Cognitive science. In S. Shapiro and D. Eckroth, editors, Encyclopedia of artificial intelligence, pages 120–124. John Wiley & Sons, 1987.

[RUP] IBM, “The Rational Unified Process. Version 2003.06.00.65”, Rational Software Corporation, 2003.

[Royce70] W. Royce, “Managing the Development of Large Software Systems”, Proceedings of the IEEE WESCON, August 1970, pp. 1-9.

[Sankey94] H. Sankey, “The Incommensurability Thesis”, Ashgate, Sydney, 1994.

[Simon83] Simon H.A.: “Models of Bounded Rationality”, (2 volumes), MIT Press, Cambridge, 1983

[Sommerville92] I. Sommerville. “Software Engineering.” Addison-Wesley, fourth edition, 1992.

[Tennenhouse00] D. Tennenhouse. “Embedding the Internet: Proactive computing”, Communications of the ACM, 43(5), 2000.

[Toffolon99] Toffolon C., Dakhili S., “Requirements Elicitation Process: An Iterative Approach based on the Spiral Model”, XVIth International Symposium on Information and Computer Sciences (ISCIS'99), Izmir, Turkey, October 1999.

[Tropos] “The Tropos Project. Requirements-Driven Development for Agent Software”. http://www.troposproject.org.

[Wautelet05a] Wautelet Y., “Multi-Agent Spiral Development of Enterprise Information Systems”, Avant-projet de thèse, avril 2005.

[Wautelet05b] Wautelet Y., Kolp M. and Achbany Y.,”S-Tropos, An Iterative SPEM-Centric Software Project Management Process”, Working Paper IAG, 2005.

[Wooldridge95] M. Wooldridge and N.R Jennings. “Intelligent agents: Theory and practice”, The knowledge Engineering Review, 10(2): 115-152, 1995.

[Zambonelli00a] F. Zambonelli, N.R. Jennings, A.Omicini, and M. Wooldridge. “Agent-Oriented Software Engineering for Internat Applications”, Coordination of Internet Agents: Models, Technologies and Applications, pages 326-346, Springer-Verlag: Heidelberg, Germany, 2000.

[Zambonelli00b] F. Zambonelli, N.R. Jennings and M. Wooldridge. “Organizational abstractions for the analysis and design of multi-agent systems.” In Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering, volume 1957 of LNCS, pages 243-252, Springer Verlag, 2000.

[Zambonelli00a] F. Zambonelli, N.R. Jennings, A.Omicini, and M. Wooldridge. “Agent-Oriented Software Engineering for Internat Applications”, Coordination of Internet Agents: Models, Technologies and Applications, pages 326-346, Springer-Verlag: Heidelberg, Germany, 2000.

[Zambonelli00b] F. Zambonelli, N.R. Jennings and M. Wooldridge. “Organizational abstractions for the analysis and design of multi-agent systems.” In Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering, volume 1957 of LNCS, pages 243-252, Springer Verlag, 2000.