Towards Knowledge Evolution in Software Engineering:
An Epistemological Approach

Yves Wautelet, Université catholique de Louvain, Belgium
Christophe Schinckus, Facultés Universitaires St-Louis, Belgium
Manuel Kolp, Université catholique de Louvain, Belgium

ABSTRACT

This article presents an epistemological reading of knowledge evolution in software engineering (SE) both within a software project and into SE theoretical frameworks principally modeling languages and software development life cycles (SDLC). The article envisages SE as an artificial science and notably points to the use of iterative development as a more adequate framework for the enterprise applications. Iterative development has become popular in SE since it allows a more efficient knowledge acquisition process especially in user intensive applications by continuous organizational modeling and requirements acquisition, early implementation and testing, modularity,... SE is by nature a human activity: analysts, designers, developers and other project managers confront their visions of the software system they are building with users' requirements. The study of software projects' actors and stakeholders using Simon's bounded rationality points to the use of an iterative development life cycle. The later, indeed, allows to better apprehend their rationality. Popper’s knowledge growth principle could at first seem suited for the analysis of the knowledge evolution in the SE field. However, this epistemology is better adapted to purely hard sciences as physics than to SE which also takes roots in human activities and by the way in social sciences. Consequently, we will nuance the vision using Lakatosian epistemology notably using his falsification principle criticism on SE as an evolving science. Finally the authors will point to adaptive rationality for a lecture of SE theorists and researchers’ rationality.

Keywords: Adaptive Rationality, Bounded Rationality, Epistemology, Knowledge Growth Theory, Requirements Engineering, Software Development Life Cycles

1 INTRODUCTION

Software engineering (SE) is by nature a human discipline. In huge software projects hundreds of analysts, designers, developers, project managers, potential users, etc. are involved in the process of creating a software application aimed to fulfill services easing the everyday work and life of thousands of individuals. To deal with such...
a process the traditional waterfall model in which requirements are collected once for all at the beginning of the project life cycle is nowadays too limited. Researches have led to the definition of advanced modeling languages as well as to more elaborated life cycles using the concept of iterative development. Their success come from the fact that requirements elicitation is no more a standalone task at the beginning of the project but rather a continuous process using feedback loops allowing to better apprehend needs on the basis of prototypes.

In this article we propose a modern epistemological reading of SE as an evolving science as well as software project knowledge evolution. As a first basis, project actors’ reasoning context is studied. Indeed, actors involved in a software project have a limited vision inherent to their bounded rationality. At first sight Popperian theories could be used to describe the evolution of software engineering theoretical frameworks. Due to the nature of SE belonging also to social sciences we will show that this vision is abusive. The Lakatosian falsification principle will be used to demonstrate the complexity of getting a crucial experiment in SE. Adaptive rationality will be used to characterize SE theorists and researchers’ expertise field vision.

This article is organized as follows. Section 2 presents the context of the research, the followed approach and justifies the utilization of the epistemologies used in the article and their combination. It also points to the article contributions. Section 3 proposes an epistemological reading of software processes and on the evolution of knowledge in SE as a discipline as well as within a particular project. To this ends we used Herbert Simon’s bounded rationality, the Popperian knowledge growth theory nuanced by the Lakatosian falsification principle and finally adaptive rationality. The implications are then overviewed in section 4 while conclusions are summarized in Section 5.

2 Problem Statement

This section presents the research context i.e. software engineering and more particularly software development life cycles; the research approach, i.e. the framework developed for applying different epistemologies at different level of a defined software development system. It then turns to the epistemological trends where the application of the particular epistemologies are justified and finally point to the main contributions of the article.

2.1 Research Context

Software Engineering (SE) is an engineering discipline concerned with all the aspects of software production. Following Arlow and Neustadt (2002), a SE methodology is made of a modeling language and a software development process.

The modeling language is the syntax made of concepts associated with visual icons used to build models (see for example the Unified Modeling Language (OMG, 2007; Rumbaugh et al. 1999; Booch et al. 1999)). The modelling language can be:

- Graphical when it uses diagrams with a couple of formalized symbols—representing concepts—linked together through relationships with eventually other graphical artefacts representing constraints;
• Textual when it uses standardized notations with parameters for computer interpretation.

The software development process defines the WHO, WHAT, WHEN and HOW dimensions of software development. A software development process can generally be split up into four disciplines (often called phases):

• The analysis discipline focuses on defining the problem (what the problem is) independently of any solution that is why it is said to be problem-oriented. Its two main purposes are:
  - To model stakeholders’ (end-users, managers, engineers, etc.) requirements by providing models allowing the software development teams and stakeholders to agree on what the system should do;
  - To build a domain model i.e. a representation of all the relevant concepts or real-world entities in a particular domain of interest.

• The design discipline is concerned with the elaboration of a solution to the problem described during analysis. It is a system-oriented discipline focusing on how to solve the problem;

• The implementation phase is concerned with the component building from scratch or by composition. On the basis of the models developed during the previous disciplines, the team should build exactly what has been requested even if there remains place for innovation and flexibility;

• The testing phase is concerned with the evaluation of software correctness, completeness, security and quality. Many approaches to software testing exist but effective testing of complex products is essentially a process of investigation rather than creating and following well defined procedure.

Following Royce (1998), a System Development Life Cycle (SDLC) is “a conceptual model used in project management to describe the stages involved in a system development project from an initial feasibility study through maintenance of the completed application”. Several SDLC have been proposed over the years. Depending on the project, some SDLC appear to be more adequate than others, best known are:

• The waterfall model (Royce, 1970) 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 (see (Boehm, 2000)) are met at the same time. Otherwise, risks are introduced in the process;

• In the incremental model the phases of the waterfall model are repeated using an iterative prototyping philosophy. The incremental process model is iterative by nature; it focuses on the delivery of an operational product with each increment. Early increments do provide capability that serves the user and also provide a basis for user evaluation;

• The spiral model is defined in (Boehm, 2000) as “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 an iterative 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.”. Using an spiral SDLC implies that risk is considered during the early stages of the project not at the late ones as in a process fully driven by sequential activities (see (Boehm, 2000)).

Modern software development processes as the Rational Unified Process (IBM, 2003), eXtreme Programming (Beck, 2005) and other Agile Modeling techniques (Ambler 2002) apply the principles overviewed into the incremental and spiral models in every day software applications development. In this perspective, the reader should keep in mind the principles of SDLCs in software development rather than the details of the nowadays used methodologies. In the rest of this article, when we will reference iterative development we point to software development in an iterative manner with no unique analysis, design, implementation and test stage but rather an iterative repetition of these stages allowing to enrich the produced artifacts on the basis of the feedback obtained by users.

2.2 Research Approach

This article defines a system to study knowledge evolution when engineering software from an epistemological point of view. The focus is thus mainly on knowledge acquisition techniques: modelling languages and software development life cycles. To be consistent with this goal, we distinguish two levels of knowledge evaluation:

- The **macro-level** is concerned with the study of the knowledge about the theoretical frameworks used to guide software development. Indeed, nowadays software developments can be very complex, involve tens of individuals and be of critical importance, facing such a context, one need advanced and mature modelling languages and development life cycles;

- The **micro-level** is concerned with the evolution of knowledge in the software project itself i.e. the knowledge on user requirements, stakeholders expectations, business processes, the organisational setting in which the system will be deployed, etc.

- **A feedback loop** i.e. each software project is another experience for testing the applied theoretical frameworks; the experience provides guidance for their evolution.

Figure 1 proposes a system view of the two levels we propose here to study knowledge evolution in SE.

In the system defined above, micro-level characteristics (the knowledge of project actors) have a strong influence on the macro-level theoretical framework (the way knowledge is collected and formalized) adoption and evolution. Consequently, section 3.1 will adopt a micro level approach and enlighten useful conclusions for the comprehension of the macro-level. The macro-level approach of section 3.2 directly envisages the evolution in the software project knowledge acquisition frameworks.
(modelling languages and development life cycles).

2.3 Epistemological Trends

The article envisages the instantiation of a series of epistemological frameworks to the micro or/and macro levels exposed above. These frameworks are:

- Herbert Simon’s bounded rationality principle: a theory addressing human problems;
- the poperian knowledge growth model: a theory addressing natural problems subject to refutation in case of empirical failure;
- Lakatosian falsification and adaptive rationality principles: theories developed for quasi empirical sciences envisaging adaptability in case of empirical failure.

(Wautelet, 2008) has developed a framework for evaluating software development life cycles of defined software engineering methodologies (applications can be found in the thesis). This framework includes both engineering and project management concepts. We choose here to adapt this framework not for evaluating a development methodology but rather an epistemology’s applicability into this particular area. Indeed, our goal is to give a genuine vision of software engineering.

Figure 1. A system for studying knowledge evolution in software engineering
using a couple of epistemological trends. To evaluate the applicability of those epistemologies onto knowledge in SE, the criteria enlighten into that framework are particularly well indicated since its goal is to evaluate a development methodology in terms of SDLC and modelling capabilities (for a full justification of those criteria see (Wautelet, 2008)). We will thus use the criteria not for the evaluation of a particular development methodology but to evaluate how a particular epistemology envisages the criteria. However the framework is a little too detailed for what we want to evaluate and we will retain the following criteria:

- **Engineering concepts.** How does the epistemology envisage a development life cycle? This will be evaluated in terms of:
  - **Process cutting:** Could the epistemology envisage a life cycle divided into phases, views, disciplines?
  - **Covered stages:** Could the epistemology envisage a life cycle not covering all of the known software development stages (analysis, design, implementation, test, etc.)?
  - **Stages scope:** Could the epistemology envisage a life cycle where stages are poorly covered?

- **Modeling language.** Does the epistemology envisage the use of a modelling language? This will be evaluated in terms of:
  - **Models representation:** Could the epistemology envisage a modelling language only representing partially of the problem?
  - **Models coverage:** Could the developed models cover only a part of process disciplines?

- **Project management concepts:** How does the epistemology envisage a project management perspective to drive software development? This criterion can be evaluated in terms of:
  - **Process description:** Could the epistemology envisage a non exact description?
  - **Project management:** Could the epistemology envisage human practices to conduct a software project?
  - **Software metrics:** What is, following the particular epistemology, the status of metrics giving a quantitative aspect of the developed software application?

Table 1 summarizes the instantiation of the criteria to the epistemologies, main lessons that can be taken when evaluating epistemologies at the light of the exposed framework are:

- Simon’s bounded rationality principle addresses human empirical problems; framework’s criteria at the light of the theory is not from particular interest since the epistemology studies human behaviour rather than scientific theories. That is why the framework is adopted at micro level—where human actors are dealing to develop a software solution—rather than at macro level—where theoretical frameworks are developed.

- Popperian knowledge growth model addresses natural problems and theories should be refuted in case of empirical failure. When applying these
software engineering criteria, the epistemology appears to be “radical” for the development process characteristics i.e. the proposed theoretical framework should be exact, capture all of the problem complexity at once, cover the whole of the known modeling and SDLC aspects and propose exact realization workflows.

- Lakatosian falsification and adaptive rationality principles address quasi-empirical problems and theories are adaptable when facing empirical failure. The criteria’s application show that Lakatos is less “radical” that Popper since the development methodology is seen as a partial guidance rather than as an exact solution.

We choose to apply Simon’s rationality principle at micro level because the theoretical framework is particularly well adapted for characterizing the human behaviour when building a software solution. The popperian epistemology is the basic reference to study science evolution especially in hard sciences, its adoption on SE was thus a first step for such a study. As will be shown in next section, its radicalism was not completely in line with the studied field that is what led us to the application of a

Table 1. Used epistemologies through SE methodologies evaluation framework

<table>
<thead>
<tr>
<th>Nature of the problem</th>
<th>Simon’s bounded rationality principle</th>
<th>Popperian knowledge growth model</th>
<th>Lakatos falsification and adaptive rationality principles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human</td>
<td>Acceptable</td>
<td>Should include all dimensions</td>
<td>Conceivable</td>
</tr>
<tr>
<td>Nature of the theory</td>
<td>N/A</td>
<td>Not subject to empirical failure</td>
<td>Adaptable in case of empirical failure</td>
</tr>
<tr>
<td>Engineering concepts</td>
<td>Process divided into disciplines</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Covered stages</td>
<td>Can be partial</td>
<td>Should be complete</td>
<td>Can be partial</td>
</tr>
<tr>
<td>Stages scope</td>
<td>Can be partial</td>
<td>Should be complete</td>
<td>Can be partial</td>
</tr>
<tr>
<td>Modeling Language</td>
<td>Models representation</td>
<td>Should be complete</td>
<td>Can be partial</td>
</tr>
<tr>
<td>Models coverage</td>
<td>Can be ad hoc</td>
<td>Should be complete</td>
<td>Can be partial</td>
</tr>
<tr>
<td>Project management</td>
<td>Process description</td>
<td>Basic</td>
<td>A convenient workflow</td>
</tr>
<tr>
<td>Concepts</td>
<td>Project management</td>
<td>Can be “ad hoc”</td>
<td>Formalized</td>
</tr>
<tr>
<td>Software metrics</td>
<td>Subject to interpretation</td>
<td>Should be exact</td>
<td>Acceptable guidance</td>
</tr>
</tbody>
</table>

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
less radical epistemology, the Lakatosian one.

We decide to apply different epistemologies, Simons’ bounded rationality on the one side, Popperian knowledge growth model and Lakatosian falsification principle and adaptive rationality on the other because (i) micro and macro levels address different kind of problems having their own characteristics so that they can hardly be studied on the basis of the same epistemology (ii) Popper and Lakatos have their own–partially divergent–vision of science evolution, the study of software engineering frameworks evolution at the light of the ideas of the two philosophers.

In a word, engineering software is a complex activity requiring a pluralist approach for an accurate evaluation. In the same line, other theories will be refereed to (for example behavioural economics) into the analysis but are considered as secondary in the context of the article so that they are not included in the table above. Finally, we preferred avoiding all new (post-modern) epistemologies (as (Lyotard 1979; Latour, 1988, Rorty, 1990)) because these are embedded in the linguistic dimension of knowledge. We decide to consider SE as quasi empirical science rather than a simple “language game”.

2.3 Contributions

Basili (1992) defines Software Engineering (SE) as “the disciplined development and evolution of software systems based upon a set of principles, technologies and processes”. These theoretical frameworks are expected to solve practical problems by proposing software solutions. SE is a practice-oriented field (where empiricism often plays an important role) and constantly evolving; however, one must dispose of a framework to build common (and preferably best) practices for software quality improvement. Kaisler (2005) points out that “We develop more experience, we not only continue to learn new practices, but we refine and hone the practices that we have already learned”. SE is the genuine discipline that emerged from this interconnection between practices and software solutions. Today’s software development has become a very complex task and no one has the required skills or time to resolve a sophisticated problem on his or her own. Software development phases need the input from lots of people having to use concepts and ideas for which they share a common understanding. This can be referred as SE’s key role: providing some common theoretical entities to allow specialists to develop software solutions.

Few articles in specialized literature point to an in depth questioning of SE knowledge evolution. As Kaisler (2005) emphasizes, the literature is mainly technical or practical and focused on the software design processes. Research methodologies, however, need to be conscientiously built to favour the development and improvement of software solutions. To this end, an epistemological analysis is of primary importance as pointed out by (Basili, 1992), “The goal is to develop the conceptual scientific foundations of software engineering upon which future researchers can build”.

The idea is to demonstrate that the roots of software development–especially those implying users intensively–can be found in social sciences and not only in exact ones with a profound consequence on the evolution towards an adequate knowledge acquisition process in software engineering. The proof of concept is based on the study of the project involved actors’ knowledge evol-
tion (micro level) towards a whole software project. Rationality evolves progressively to them into enterprise software projects, so that an iterative development life cycle–by nature including this postulate–is better adapted for this type of developments. On the same way, SE theoretical frameworks (macro-level) evolutions especially in the field of modelling languages and development life cycles also evolve progressively by adapting existing ones rather than by defining completely new approaches; micro and macro levels are envisaged in the form of a system.

This study is a contribution to the philosophy of computer science and like most philosophical treatises it neither addresses itself to any practical problems, nor does it solve them. It however provides a new (nuanced) vision of a well known discipline: SE as a quasi-empirical discipline rather than a hard science. This is an interesting reference for researchers and other SE practitioners (see section 4, implications).

3 A STUDY OF KNOWLEDGE EVOLUTION IN SE

This section details the practical application of different epistemologies on the micro and macro levels defined earlier.

3.1. Micro-Level Approach: A Study of Project Actors’ Rationality

This section evaluates knowledge evolution into software projects (micro level) at the light of Simon’s bounded rationality principle, conclusions are then brought to the development life cycles (macro level).

3.1.1. Bounded and Perfect Rationality or the Quest for a Satisfactory Solution

Herbert Simon (Nobel Prize in economics, 1978) proposes the concept of bounded rationality (Simon, 1983) to characterize the human reasoning and behavior in uncertain situations. This concept illustrates the idea that human beings have limited abilities to analyze all the parameters and implications of the solutions they could deploy to a given problem they face. According to Simon, the rationality of human beings is related to the present and to the psychological biases. The rationality suggested by Simon is much broader than the perfect rationality which would allow one to be able to evaluate all the present and future parameters of a particular solution. This idea of perfect rationality is often used in economics and whereas the concept of bounded is more often used in management sciences, Simon has won a Nobel Prize for his work.

The concept of perfect rationality is extremely useful to solve some theoretical problems (Arthur, 1994). Perfect rationality presupposes that individuals have much higher computational abilities than they have in the real world. Indeed, beyond a defined 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 (this leads by bounded rationality). So human behavior 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 that the complex interactive situations can be seen as undefined problems. To face such problems, Arthur (1994), in accordance with the behavioral economics (and with the bounded rationality principle), proposes to think the human mind as a “problems simplification machine”. 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. The principle of bounded rationality results from a particular conservative incrementalism materialized in a careful attitude always concerned by feedbacks provided by human interactions (Kaplan & Kaplan, 1982).

This vision of rationality can be used to characterize an organization behavior as a team involved in the process of engineering a software product. Indeed, 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 rationally 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. The illustration on Figure 2 shows the multiple actors’ visions of a software product. When building a software solution several actors (and models) bounded rationalities intervene, for instance:

- when modeling the organizational setting:
  - **stakeholders** as, for example, **engineers** explain their understanding of the organization and other business processes;
  - **software modelers** represent what they understand of the business processes using a **modeling language** containing its own limitations.

- when eliciting and modeling user requirements:
  - **end users** explain their expectations of what the system should do;
  - **software modelers** represent what they understand of end user requirements using a **modeling language** containing its own limitations.

- when designing the software application, **software designers** build design models (using a **modeling language** containing its own limitations) on the basis of what they understand of the provided analysis models and other artifacts produced by analysts.

- when implementing the system, **developers** produce a software system (using **programming languages** containing their own limitations) on the basis of what they understand of the provided design models and other artifacts.

Each actor involved in the software development process has to deal with:
his own rationality which is by essence bounded;
the bounded rationality of the other actors he works with;
the inherent limitations of the modeling languages he has to deal with (models are by essence limited and do not allow to capture the whole of the reality of the problem, see (Rum- baugh et al., 1999).

The bounded rationality principle is in accordance with the rationality used in social sciences. This concept has allowed the emergence of behavioral economics which tend to be more and more dominant. Using such a vision implies to take into account that human beings cannot evaluate perfectly the moving contexts and that this misunderstanding of the situation leads them to seek a satisfactory solution. This satisfactory solution depends directly on each individual and the quest for a satisfactory solution allows people to meet their fundamental needs and to develop themselves. In this human reasoning model, no way exists 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.

3.1.2. Actors Rationality and SE Methodologies

The bounded rationality principle is an interesting concept for the theoretical analysis of software development. It allows characterizing the complexity observed in software solutions to give an efficient response to organizational problems by nature hard to define. Organizational interactions allow project stakeholders to improve their knowledge in an iterative manner and to improve their requirements comprehension. That is why, facing such a context, one need an accurate modeling language as well as a development process designed to favor stakeholders’ knowledge evolution and allowing to constantly adapt the software product with those new refinements. Iterative software development where requirements are collected and revised progressively on the basis of early testing to get the best adequacy between stakeholders’ knowledge and the developed products is better in line with actors’ bounded rationality. Through the use of a process better in line with human rationality as the iterative one, users’ perceived quality and adequacy of the software solution can then be improved.

3.2. Macro-Level Approach: Lakatosian Lecture of an Artificial Science

In this section we will present the Popperian knowledge growth model applied to SE developments at macro-level (a micro-level application is also briefly done but is secondary to the demonstration) and nuance this application with the Lakatosian falsification principle. We will then emphasize that a Lakatosian epistemology is directly in line with the notion of bounded rationality evoked in the previous section.

3.2.1. The Knowledge Growth Theory: An Application to SE Methodologies

According to Popper (1972), scientific knowledge evolves thanks to a continuous and infinite problem 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, theorize and solve. The empirical refutation of a proposed theory induces the emergence of new theories subject to empirical refutation creating other 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” (Popper, 1959) any theory refutation allowing the emergence of new problems. This refutation constitutes a source of inspiration for the development of new theories supposed to solve the new problems.

Strictly speaking Popperian epistemology has to be applied onto the macro level of the defined system. Indeed, SE methodologies, the theoretical frameworks envisaged into this article have, following Popper, to be subject to falsification through a crucial experiment. This vision is, following us, not really adapted to study the evolution towards the nowadays SE methodologies using advanced modeling languages and mature SDLCs since the process is more evolutionary and not subject to direct
refutation. This is studied in details into the next section.

Before continuing the macro-level analysis, a parenthesis is done here and we apply the Popperian knowledge growth theory to the micro level to evaluate if this could have consequences on the theoretical frameworks at macro-level. In that perspective, according to (Toffolon & Dakhili, 1999), the Popperian knowledge growth model can be used to justify the better performance of iterative software development (in terms of matching with user requirements). 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 exist and, moreover, an iterative process leads to a moving evolution of this context. Stakeholders, by using a software solution, modify their needs 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 when they approach the software system. Consequently their requirements evolve iteratively during the software development. Toffolon and Dakhili (1999) explain that this evolution can be characterized by a Popperian lecture. This consequently points, at macro-level, to the adoption of a SDLC envisaging the project knowledge evolution in an iterative manner.

### 3.2.2. Criticize of the Falsification Principle at the Light of SE methodologies Evolution

In his work about the mathematical formalisms, Imre Lakatos proposes two categories of theories: Euclidean and quasi-empirical theories. Whereas the firsts 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. Indeed, a quasi-empirical science can be tested empirically but remains specific to a particular context that explains the “quasi” diminutive. In relation to Simon’s works, all artificial sciences can be seen as quasi-empirical sciences but the inverse is not necessary true (Gianluigi, 2006). As we will show in this section, this point of view is different from the Popperian one because in his epistemology, theoreticians have to change the whole theory if it is falsified. We think that the Lakatosian epistemology about quasi empirical theories is useful for the study of knowledge evolution in the SE science (at macro-level) because the process of theories adjustment he proposed is directly in line with the quasi-empirical logic observed in this field. The aim of this section is to moderate the use of Popperian epistemology at macro-level. At first sight, the Popperian knowledge growth model is very close to what Lakatos calls the “quasi-empirical theories”. The two visions are however different and, according to us, only the Lakatosian is adequate for a study of SE methodologies evolution.

Popper developed his epistemology for “hard sciences”, especially physics. Science
is mainly considered in terms of “representation of the world” rather than in terms of “efficient answers” as it would be in the field of applied ones. This detail is of primary importance since, at the opposite of hard sciences, a universal framework performing at best in every software development cannot exist. For showing this we will briefly refer to the micro level. Indeed, as it has been shown previously there is no universal and optimal software solution: no development is identical to one another since each organization is different. Contrary to hard sciences which describe the “reality” as it is, SE incorporates in its processes individuals’ way of solving specific problems. The practical experiment 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”. Indeed, each new project is a new experiment based on various theoretical concepts (modeling languages, development life cycles, programming languages, etc.) and trying to satisfy various requirements. Each software solution is developed for a particular organization employing various people pursuing various goals and it is very difficult to compare with one another so that the reasons of an empirical failure are hard to determine. Moreover, in software development it is very hard to correctly identify the nature of the error (Priestley, 2005). The reason(s) 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 at physical layer. The variety of errors strongly complicates the idea of a “crucial experiment”. We can consequently not, in case of empirical failure, reject all the applied SE theoretical bases as for instance the methodology used within that project. Theoretical tools can be very powerful and particularly well indicated in the context of the “failed” development but the reason of failure could be exterior to them. On the basis of those overviewed characteristics of software development we categorize SE as a “quasi empirical applied discipline”.

The falsification principle specific to the Popperian epistemology can consequently not be applied to the quasi-empirical (or artificial) discipline of software engineering as it would be to disciplines of hard sciences. This subscribes the fact that the nature of SE can also be found in social sciences rather than only in hard ones. For sure, the empirical failure of a software solution has to lead to calling into question the SE theoretical foundations but they cannot be directly and purely rejected. The Popperian model of knowledge (also called “dogmatic falsificationism” in philosophy of science) appears to be too radical for an epistemological reading of SE theoretical frameworks evolution. This Popperian radicality has already been recognized and caused a plentiful literature in philosophy of science (see e.g., (Kuhn, 1996) or (Lakatos, 1977)). In line with the opposition to the radicalism of falsification, Lakatos developed its methodology of “research programme” (see (Lakatos 1977) for the theoretical basis and (Wautelet et al. 2008) for an application on object and agent orientation).

3.2.3. Software Engineering Research at the Light of Adaptive Rationality

The Lakatosian epistemology (often “called methodological falsificationism”) 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” 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 programme in which the theory has been developed), nothing obliges the theorists to modify their theoretical framework. This vision is particularly well adapted to an epistemological reading of SE.

The Lakatosian epistemology is based on the notion of adaptive rationality which could be seen as the result of Simon’s bounded rationality. Indeed, theoreticians do not reject a theory when it is falsified because they will try to improve it by developing some new solutions. Thanks to this process, the descriptive dimension of theory will be improved. No theory can claim describing directly and perfectly the “reality” because theoreticians (SE researchers) are governed by a bounded rationality leading them to a continuous knowledge acquisition process along all the software projects they are facing. Their knowledge grows progressively in an iterative manner as it is for the knowledge of the people involved onto a software project. A kind of parallel can here be drawn between the macro and micro levels even if the approach of those two types of actors is rather different and can therefore not be directly compared. In line with Lakatos, Simon is opposed to the Popperian falsificationism because it implies to ask whether theoreticians are perfectly rational or not (Simon, 1991). According to Simon (1991), theoreticians have always good reasons to do what they do; once they are governed by bounded rationality and they are just unable to describe directly and totally the “reality” like the falsificationism presupposes it. The bounded rationality developed by Simon can be seen as a justification of the Lakatosian epistemology (LeMoigne, 1994).

The adaptive character of the human mind results from a bounded rationality. This iterative vision of knowledge described by Lakatos is directly in line with the evolution of SE theoretical frameworks (for example from Merise to UML, from waterfall to iterative development, etc.). By solving specific organizational problems, SE can be considered as an “artificial science” that can better be analyzed using a Lakatosian perspective than a Popperian one.

4 IMPLICATIONS

Everyday work implications of the study of knowledge evolution into software projects and onto development life cycles gave several interesting conclusions important in the daily practice of researchers and
other practitioners. Figure 3 summarizes the conclusions taken from our epistemological analysis.

Our epistemological approach of the micro level defined earlier in the article points out that:

- the ability to express requirements and to understand their nature is by essence limited since actors are dealing with bounded rationality representing problems and designing software solutions with by essence models so that this process cannot be proceeded once for all into the software project but should rather be continuous;
- knowledge grows within human organizations. When developing software solutions, knowledge about user requirements and further expectations grow during the project life cycle. Previously identified requirements should not be refuted but rather refined continuously at the light of users’ feedbacks.

Iterative development allows to better deal with actors’ bounded rationality and with the knowledge evolution into the software project thanks to the feedback loop on the produced solutions it offers (micro level feedback loop).

At macro-level, the most important aspect that can be deduced from our work is that SE does not belong to hard sciences but can rather be considered as a quasi-empirical discipline as defined by Lakatos. Theoretical frameworks produced by SE researchers are subject to evolutions and adaptations since they are dealing with adaptive rationality (macro level feedback loop). Evolution in SDLCs frameworks

Figure 3. Software engineering's systematic approach: main epistemological analysis conclusions
has nowadays led to iterative development life cycles as overviewed in section 2 for some of the reasons evoked above but the framework will evolve by adoption of further evolutions.

Some derived conclusions can also be pointed out for software engineering researchers. Indeed, due to the fact that with a Lakatosian reading of SE knowledge evolution no crucial experiment can reasonably be established, empirical testing of the developed SE theories should be realized on case studies but the quantitative burden of the proof of their better performance can hardly be established. This consequence is notably useful for researchers working on the elaboration of new SE methodologies. Their experimentation aimed to establish the better performance of their development models/processes/frameworks should be as most as possible quantified but since this cannot constitute an unambiguous proof, it must be complementary to a qualitative typology.

5 CONCLUSION

User requirements that have been poorly taken into account as well as modeling and development languages inspired by programming concepts as opposed to organization and enterprise ones has led to a software crisis. Solutions can be found at different levels, and among those conceptual modeling and iterative development have been noticeable evolutions. Indeed, proceeding iteratively allows, thanks to continuous feedback loops, to develop a software product having a perceived quality in better accordance to users expectations.

This article has presented an epistemological reading of knowledge evolution in SE at both the level of software projects (micro level) and at the level of SE frameworks (macro level). At micro level, we studied knowledge evolution into a software project involving tens of actors governed by bounded rationality. At macro level, the article concludes SE is a quasi-empirical discipline; indeed due to the impossibility of a crucial experiment, upcoming innovations can be based on refinements of existing theories and do not induce a complete refutation of existing techniques. SE theorists and researchers are governed by an adaptive rationality. The article concludes and advocates that on the basis of our epistemological lecture, the use of iterative development is better adapted to drive SE projects due to knowledge evolution and with bounded rational actors and SE theoretical frameworks evolve in an adaptable manner. SE can be considered as a quasi-empirical discipline rather than a hard science. Finally, the article constitutes a contribution to computer science philosophy.

More work should be carried out in SE fundamentals by studying other methodological frameworks and epistemological foundations to provide the computer science researcher a more accurate vision of the research field that he or she is working on. Other works showed that aspects such as object and agent-orientation are also strongly in line with Lakatosian theories. An interesting point would be to study agent software entities at the light of Simon’s theories.

REFERENCES


ENDNOTES

1 In the context of this article, SE methodology and SE development process are considered as synonyms, when using these terms we refer to a modeling language and a software development life cycle.

2 Daniel Kahneman psychology professor at Princeton won, in 2002, the Nobel Price in economics for having integrated insights from psychological research into economic science, especially concerning human judgment and decision-making under uncertainty (see (Kahneman, 2002)).

3 Herbert Simon’s work is structured around three themes [Parthenay05]: bounded rationality, organizational processes and epistemology of artificial sciences. Bounded rationality and organizational processes were presented in section 3.2 as an accurate conceptual framework to describe project actors’ rationality. Simon defines Artificial Sciences as disciplines constructed by human mind to solve specific problems at a specific time [Simon69]. The artificial character refers to the contingent nature of the solution.

Ad hoc is a latin expression which means “for this purpose”. In philosophy of 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 initial theory. Popper rejects this “ad hoc” character in science and, following him, this should be considered as a kind of intellectual dishonesty (see (Popper, 1959)).

4 (Lakatos 1977) reminds that science is a human practice and that it is sometimes very difficult for scientists to reject a theoretical framework when they have devoted several years of their life working on and developing it.

Yves Wautelet is an IT project manager and a postdoc fellow at the Université catholique de Louvain, Belgium. He completed a PhD thesis focusing on project and risk management issues in large enterprise software design. Dr. Wautelet also holds a bachelor and master in management sciences as well as a master in Information Systems. His research interests include aspects of software engineering such as requirements engineering, project management, software development life cycles, component-based development, epistemological foundations of SE and CASE-Tools but also information systems strategy.
Christophe Schinckus is a postdoc fellow at the University of Québec at Montréal. He completed his PhD in financial economics at the University of Paris I Panthéon-Sorbonne. Economist and philosopher of science, Dr. Schinckus works on epistemological analysis of financial theory and especially econophysics. He has also a BSc in computer science and is interested in software engineering processes.

Manuel Kolp is a professor in information systems at the Université catholique de Louvain, Belgium where he is also head of the Information Systems Research Unit and Academic Secretary of Research for the Louvain School of Management. Dr. Kolp is also invited professor with the Universitary Faculties St. Louis of Brussels. His research work deals with agent-oriented and socio-technical architectures for e-business and ERP II systems. He was previously a Post Doctoral Fellow and an adjunct professor at the University of Toronto. He has been involved in the organization committee of international conferences and has chaired different workshops. His publications include more than 70 international refereed journals or periodicals and proceedings papers as well as three books.