

Mapping Agile Methods to ERP: Directions and Limitations

Rogério Atem de Carvalho¹, Björn Johansson², Rodrigo Soares Manhaes³
Instituto Federal Fluminense, Brazil, ratem@iff.edu.br

Copenhagen Business School, Denmark, & Lund University, Sweden, bj.caict@cbs.dk
Universidade Estadual do Norte Fluminense, Brazil, rod@uenf.br

Abstract

Historically Enterprise Resource Planning (ERP) systems have been developed in a waterfall-like approach, which has been resulting in that when they are finally deployed in the adopter organization often ends with misalignment problems between its delivered functionalities and the business requirements the organization would like to have supported. The way for organizations to deal with the problem of misalignment is to customize the ERP system to a high extent. However, the proposals for minimizing misalignment problems are mostly focused on the same traditional, heavyweight waterfall-like approaches, therefore creating a loop where requirements are identified long before they are implemented, going back to misalignment problem. On the other side, the last decade has witnessed the rise and growth of agile methods, which have both close communication and fast response to changes among their main values. This paper aims to provide a mapping between the main agile methods and ERP development processes, as well as to present and discuss directions and limitations to this approach.

Keywords: Agile methods, enterprise resource planning, software development

Introduction

The ERP environment faces constant changes and reassessment of organizational processes and technology, therefore project management methods used within ERP deployment must provide adaptability and agility to support this evolutionary environment in which ERP customization takes place. Alleman (2002) and Meszaros and Aston (2007) agree that the application of scientific management – here represented by “high-ceremony” or document-driven processes – is understandable in many ERP deployment cases since many target companies has a tradition in engineering and business that make them think in terms of waterfall development processes. However, Alleman (2002) alerts that “the use of predictive strategies in this environment is inappropriate as well as ineffective since they do not address the emergent and sometimes chaotic behaviors of the market place, the stakeholders, and the vendors’ offerings.”

In the context of ERP customization, a well-known problem is the misalignment or misfit between the ERP functionalities and the business requirements (Soh et al., 2000). It can be claimed that this problem is caused mainly by communication bottlenecks and difficulties on responding to changes on requirements (Johansson & Carvalho, 2009). Traditionally, heavyweight waterfall-like approaches are applied to try to minimize the misalignment problem, with results that are still unsatisfactory. On the other side, during recent years we have witnessed the rise and adoption of agile methods, having both close communication and fast response to changes among their main values. This paper maps some of the main agile methods to ERP development and presents some limitations and future directions on agile methods as an approach for ERP development.

Agile Methods and ERP Development

Johansson and Carvalho (2009) suggest that the misalignment problem is related to requirements management processes used in ERP development. In fact, communication between business people and development teams to a high extent occurs in a waterfall-like approach, which creates a bureaucratic environment where communication is done by documents and not directly between people. Besides that, the ever-changing globalized business environment makes requirement changes could be seen as “natural” events. Therefore, it is necessary to have a development process that simultaneously enhances communication between ERP stakeholders and copes with changes. Another angle of this problem is that information systems, such as ERPs, deliver too much functionality, which overload users with information. The information overload results in that even if the system contains needed functionality, user may not know about it and have a hard time to find as well as understand how to use the specific functionality. This indicates that instead of talking about misalignment a better way of describing what is needed could be synchronization between stakeholders, or more specifically, business synchronization. From this it can be said that a focus on business synchronization in the form of describing how ERPs as well as organizations could be developed in tandem would be fruitful. This implies that a focus on how to achieve efficient communication between developers and users, and fast response to change is needed.

A way of both achieving an efficient communication and fast response to change is to deliver software in small and frequent increments, as advocated by agile software development methods. Agile methods can be said are a reflection of Lean Manufacturing techniques in the software engineering realm. Sutton (1996) pointed out that Lean can be a good middle ground between craft development and “software factories”, stating that use of lean techniques lead to both quality improvement and cost reduction. Later, Raman (1998) argued that lean software development is a feasible methodology for software development. It is important to note that the term “Lean Development” (LD) is becoming more used nowadays, however the authors understand, in concordance to Fowler (2008), that agile methods are conceptually aligned to Lean Manufacturing and adopt a series of its concepts. Agile methods are well accepted in general by the software development community, however, there is no evidence that they are in use in the ERP development community.

Agile development is a generic label for all methodologies based on Agile Manifesto values (Beck et al., 2001). This manifest is a document recognizing four values and twelve principles that basically points out: close collaboration and proximity between the development team and the stakeholders; recognizing requirement changes as results of learning and opportunities to achieve competitive advantage for the organization; the search for simplicity and elimination of unnecessary documentation and processes; emphasizing people rather than processes; and establishment of delivering business value to the stakeholders as the ultimate goal and the ultimate measure of progress.

Fowler (2005) lists two fundamental differences between agile and traditional methods to software development. First, agile methods are adaptive rather than predictive. While predictive methods see change as a problem to be avoided and an undesirable and costly deviation from the plan, adaptive methods embrace change as an opportunity for improvement. Second, agile methods are people-oriented, not process-oriented. In the agile view, processes are intended to support people's work rather than to be strictly followed. This leads to recognizing software developers as self-organizing “responsible professionals” and not as disposable and interchangeable “plug compatible programming units”.

Unlike traditional methods, agile methods do not make comprehensive up-front planning. From an agile point of view, planning is a “constant process of reevaluation and course-correction throughout the lifecycle of the project” (Beck and Fowler, 2000). Development is a learning process, an exercise in discovery (Poppendieck and Poppendieck, 2003), and constant planning is the key point for achieving a process that accommodates learning. However, it is impossible to improve learning if the development team has few or no contact with stakeholders, which are the ultimate requirements specification producers. In fact, agile methods recognize that developers and stakeholders must have close cooperation, so development activities are based on concrete feedback rather than speculation.

Agile methods use short iterations, which means that in short periods of time, working software is released so that the stakeholders can check if the implemented software is exactly what they need. For each iteration is allocated a small number of features, which are implemented by the team within the iteration. The iterations must be as short as possible. This means that the software is delivered into frequent and small increments. This technique is common to all iterative and incremental lifecycles, and is used to cope with not well established or instable requirements. In that way, it is possible for both developers and users to explore incrementally the requirements and the way the software is built, allowing a better adherence to the users’ demands. Furthermore, smaller increments and shorter iterations, faster feedback cycles, leads to a very fine granularity project control and allows rapid correction of deviations from business needs (Martin, 1999).

In the same way, the requirements elicitation process should also be incremental. The features are not detailed at the beginning of the project, but only a list of features is initially extracted. In this context, each item from this list is not a requirement, but a record of its existence, a commitment for future conversation (Jeffries et al., 2000). The details about a feature are only analyzed in depth within the iteration which the feature is allocated.

Currently, the most widely adopted agile methods are Scrum (Schwaber, 2004) and Extreme Programming (Beck, 1999; Beck and Andres, 2004). Scrum is a management-centered approach characterized by time-boxed iterations called *sprints*, project tracking by feature lists called *backlogs*, diary meetings and iteration periodic retrospectives for continuous improvement. Scrum do not prescribe engineering practices, the team must choose the techniques that best adapt to its environment. Extreme Programming (XP) is an approach that strongly supports the development team at the engineering point of view. XP is built around a set of values, principles and practices. The values are the philosophical background, the practices are the concrete expressions of the values and principles are guidelines to translate the values into practices. XP's practices focus on very small iterations (1-2 weeks), all-time peer review, test-first programming, refactoring, emergent design and continuous integration. In the real world, several teams use both Scrum for project management and XP for engineering issues (Kniberg, 2007), though XP itself also covers management, planning and so on.

In the next section the main agile-related practices are discussed in relation to ERP development. It is important to note that some of them are directly related to agile methods, while others were embraced by these methods. Since the distinction of techniques’ origins is not in the scope of this paper, they are treated equally in this text.

Mapping Agile Methods to ERP Development

In this section the mapping of the main agile practices to the ERP realm is done, by presenting their principles and suggesting how they can be used in this specific environment.

Iterative and Incremental Development

Developing an ERP incrementally conflicts with the common sense in the Enterprise Information Systems community of the necessity of modeling the organization's processes, manufacturing resources and structure, as suggested by Vernadat (2002) and Shen et al (2004), in advance to the system development. In fact, modeling beforehand is necessary to, among other things, to identify possible process improvements and integration points, however, this creates a waterfall-like development process, bringing all the well known problems related to this kind of process.

Therefore, one question to be solved when using agile methods (or any iterative incremental lifecycle in general) for developing ERPs, is how to both achieve (a) incremental requirements elicitation and (b) light rework due to late process integration identification. This second situation, which occurs when, during the system development, a given business process is implemented into the software without identifying all its connections with other process, or even as it was a standalone process – a natural situation in a process where the requirements are identified incrementally. Thus, what happens in later iterations, when requirements are better understood, is the realization that this requirement needs to be integrated to other requirements, leading to changes in code and consequent rework.

This situation can be described as the “Late Integration Problem”, which will occur every time the whole business process modeling not is done before the actual development of the system takes place. Moreover, this problem can be generalized to how to cope with constant changes in business processes, without incurring in excessive rework.

The answer to this problem is that the ERP development framework must provide fast and flexible process integration – preferably through configuration instead of coding, so that integration points identified late not will demand heavy rework. In that way it is possible to rapidly reconfigure and integrate business processes so that both incremental requirements elicitation and changes in requirements are dealt with. For achieving this, the framework must provide ways to rapidly connect business process while keeping loose coupling, enabling the automatic – or configurable - creation of proper calling chains, GUI navigation, and information passing between processes. With these features it is possible to identify late integration points and rapidly provide integration among processes.

Ubiquitous Language

The concept of *ubiquitous language* is very important for communication, feedback and close collaboration between a customers and developers (Evans, 2004). When communication between customer and the development team uses less documents and textual specifications and goes more too face-to-face dialogue and learning, there is a need for an effective shared language between those groups. The ubiquitous language is a common language, covering the entire communication chain from customer and business analysis to the team's internal conversations and coding. The ubiquitous language reflects and provides feedback about the domain and must be exercised all the time in all communication tasks. This language does not necessarily uses the exact customer jargon, but an unambiguous and contradiction-free version of the domain knowledge. A ubiquitous language is not built in a single step, but it is iteratively refined and improved. According to Evans (2004), the model has to be used “as a backbone of a language”, so “a change in the ubiquitous language is a change to the model”.

Creating a ubiquitous language is extremely important for making communication between developers and users more effective. How this can be done in the ERP system case to a high extent depends on the ERP business model. Grudin (1991) describes this

problem by identifying when and how the users are identified in the development process. He distinguishes between three different development situations: competitively bid/contract development, product development, and in-house/custom development. The difference between these three is that, when it comes to when users are identified, the product development situation first identifies the users when the system is delivered.

Test-Driven Development

Test-driven development (TDD) is a technique that consists of writing test cases for any programming task (feature creation or adaptation, improvements, bug corrections etc), before these implementations are performed. Koskela (2007) said TDD is intended for “solving the right problem right”, meaning to achieve the correct solution that exactly matches the business problem.

TDD is not only a set of testing tools, but, primarily, a design technique, and in TDD design are made all the time (Beck and Andres, 2004), having no place for a “design phase”. In the TDD approach, the current design is only suitable in the current iteration, being the design improved for accommodating new requirements within each iteration. With good programming and design techniques, the design of a system can be continuously improved without falling into the famous Boehm's cost of change curve (Boehm, 1981). Modern programming techniques, powerful development tools, more expressive and simple languages, more powerful computing and other advances make the cost of change increase very slowly through the time for the most features. Poppendieck and Poppendieck (2003) pointed out that there are a few “high-stake constraints” such as the choice of programming language and architectural layering decisions which could have a high cost of change later on. All other, largely the most, follow a horizontally asymptotic cost of change curve.

TDD is directly applicable to ERP development, in special when dealing with high-level (black-box) testing. Given that an ERP will either substitute an existing system (or collection of systems) or will automate business processes, plenty of test cases will be available. However, the presence of testing data is not enough. Since ERP is both integrated software and an adaptable framework, regression testing is a must. Regression testing occurs, for instance, when a given module is altered to meet a given customer's specific needs. After proceeding with the unit tests, it is necessary to run integration tests to ensure that modifications do not create side effects on related parts of the system. Although this seems to be obvious, one of TDD tenets is that automated test are built in an incremental way, so when the system becomes more complex, testing also becomes more complex, but if the tests follow this complexity growing, they will cope with heavy testing jobs. Therefore, a possible difference between current ERP customization and this technique is that besides supplying the necessary information for changing the system code – including the code itself or proper APIs, it is necessary also to supply all the test scripts, so that integration also can be tested. Moreover, when new updates are provided to all users, customers that have implemented exclusive features must (a) re-implement these features on the upgraded version and (b) run all the tests against the new version of the system.

Continuous Integration

In many projects, integration brings up a series of problems that needs to be taken into account: incompatibility between modules, broken dependencies, out of date modules, low test coverage, lack of compliance to coding standards etc. This is the so-called “integration

hell” which implies that the greater the amount of code to integrate is, the more complex is the integration process, as described by Fowler (2006).

Continuous integration (CI) is a practice proposed to solve the integration problems by providing both agility and fast feedback, consisting of making (at least) daily project builds (Fowler, 2006). The concept of build used here is the whole process comprising: get the latest source code from the main repository, build the application and the database, perform both testing, inspection and deploying, run software in a production similar environment and give automatic and real-time feedback to developers, ensuring that the integrated parts act as a cohesive whole and that there is always a working version of the software available in the repository.

The environment needed for CI to work can be from a simple script to complex, full-featured continuous integration servers. Though CI seems to be an infrastructural topic, it is primarily a “shared agreement by the team” so that they keep the source code always passing on all tests and inspections and running the build process every few hours (Shore, 2005).

The main goals of continuous integration are improving quality and reducing risk (Duvall et al., 2007). Quality improvement comes from the execution of all tests, code inspections and deployments for each build. This assures that overall software is always on a valid state and prevents the system to be inadvertently broken by a developer's code commit. The risk reduction is provided by the high frequency of builds and the fast feedback mechanisms. CI immediately warn the developers if something goes wrong, usually by e-mail, but even by SMS messages or visual warnings.

This technique can bring a series of benefits for ERP development involving code changes. First, CI can help to continuously provide working versions of a customized module, which can in an incremental way be checked by users as modifications and new features are introduced, , reducing the risks of occurring misfits. To achieve such benefit, it is necessary that key users stay near the development team, testing the increments and providing constant feedback. In fact, using CI automated tests and expressive code – that provides direct mapping between requirements and code can automate part of the validation tasks, reducing the effort of key users, and providing very fast feedback.

An ERP provider and/or its partner network must keep not only a CI environment but also all configuration and test cases, both the ones used for standardized deployments, and also the customized ones. This environment can be recovered at the time of any future upgrade, and will help when a new upgrade comes and the modifications of specific customizations must be re-inserted into the upgraded module.

Emergent Design

In an iterative and incremental, agile-style lifecycle, design is not performed up-front, but is incrementally evolved as it meets the current iteration requirements. Any feature that can be useful to future iterations must be designed in future iterations, which avoids both speculation and resource wasting and keeps the team focused on designing the features prioritized by the stakeholders.

Bain (2008) defines emergent design (ED) as “a process of evolving systems in response to changing requirements, better understanding of existing requirements, and in response to new opportunities that arise from new technology, better ideas, and a changing world”. To be able to follow this principle, some disciplines must be applied, such as TDD, refactoring, expressive code and heavy use of design patterns.

When it comes to ED and ERP development it can be stated that this technique is limited, given that in any ERP environment (which also can be said is a fact in any

framework based environment), ED has limitations in refactoring deepness, since the core framework cannot be changed *ad hoc*. One restriction could be access to the ERP source code which is directly related to the basic architecture of the ERP. For instance, in the SAP case there is the use of advanced business application programming (ABAP) which probably restricts the possibilities to implement an ED approach. From that it could be said maybe ED is more applicable to the creation of entire new ERP modules, when programmers have access to all design decisions in this particular piece of the system. Also, it is highly applicable in in-house and open source ERP development, but also in some proprietary ERP systems such as Microsoft Dynamics Navision that provides its end-user with total accessibility to the source code.

Agile Methods Limitations in ERP Development

The mapping of agile methods to the ERP development realm presents cultural and technical limitations, while some of them also exist in the context of other types of systems; others are very specific to ERP customization and life cycles:

Cultural Limitations

- Scientific Management and its predictive planning approach are well disseminated and work well in other types of projects, making it hard for high management to accept following a reactive way of managing the ERP customization and deployment project.
- Many ERP, such as SAP, uses document-driven processes to make clear separated roles communicate. This type of process not only makes communication slower, but also stimulates the creation of modeling and communication artifacts that do not aggregate value to final users. Besides that, stimulates the high specialization of the workforce, making people less flexible in skills, going against one of the modern production principles of flexible production resources. Flexible resources makes adaptation to fluctuations (such as in specific skills during the project) easier and cheaper, reducing resource bottlenecks and allowing higher degrees of product customization.

Technical Limitations

- Pioneer ERPs pay the price of legacy technologies and their limitations, which sometimes becomes a burden for the use of fully object oriented techniques, in which Agile methods rely. Moreover, built-in tools, such as SAP's ABAP, on one hand simplify some development tasks, but on the other, make the whole ERP development community depend on a single supplier to evolve these tools - while ERPs based on open languages and standards take advantage of all language and tool evolution, independently of the ERP supplier.
- The database-centric ERP configuration task brings all the limitations of using RDBMS, including the difficulty of using agile techniques such as Refactoring and Emergent Design (Ambler, 2008).
- A limitation to agile-based ERP customization can be in the ubiquitous language, if the ERP, because of eventual architectural idiosyncrasies, does not allow a one-to-one mapping between business domain and domain model. In some cases, uncoupling techniques as façades (Gamma et. al, 1995) or even an anticorruption layer (Evans, 2004) can be useful. These techniques, however, does not eliminate the duplicity of languages (non-compliant to UL in the ERP itself and compliant in the customization).

Conclusions and Research Directions

The statistics on unsuccessful ERP implementations urge for new ways of facing the misalignment problem. It can be concluded that agile methods in ERP development can achieve efficient communication between developers and ERP end-users if some limitations can be overcome.

This paper started on the premise that the adopter has already chosen a specific ERP and already decided to change part of its code to make it fit better to the adopter's business needs. Although this situation seems to be somehow a paradox for the ERP world, it is more common than it should be. Additionally, considering new ways of providing and developing ERPs, such as Open Source ERPs, it can be stated that the role that end-users have in development have changed, indicating that they are more involved in the development and in that way it could be a change towards agile methods.

This paper contributes by providing a mapping between the specific techniques used by agile methods and ERP development, complementing Meszaros and Aston (2007), who describe the creation of a development environment and the cultural changes needed for agile development for a specific ERP product; and Alleman (2002), who describes the decision process involved in customizing and deploying agile methods in ERP development.

It can be concluded that agile methods in development of ERPs builds on the premises that efficient communication between developers and ERP end-users can be achieved. One way that agile methods advocates how this could be achieved is by delivering software in small and frequent increments. However, interesting research questions is then how efficient communication can be gained in different kinds of business models used to deliver the ERP system - Open Source ERP, In-house Development (based on open source or not), External Hosting ERP, Direct Sale ERP (the vendor sales directly to the customer) or Indirect Sale ERP (the vendor uses a partner channel as distributor), since these models differs a lot on how communication can be done between developer and end-user.

It can also be concluded that a potential benefit in agile development of ERPs, which is especially relevant for ERP customization, is the fact that agile methods are adaptive rather than predictive. This means that if it is possible to implement them in ERP development, it can be stated that it is more likely that the ERP and the organizations' business processes will develop in tandem and thereby fulfill the goals of business synchronization. The main conclusion is that agile methods for ERP development have a great potential but it demands a close cooperation between different ERP stakeholders, so that customization is based on concrete feedback and not speculation.

A future outcome of this work is to identify agile and lean practices adoption level in ERP development, differentiating between proprietary, free/open source, and in-house solutions. Also, investigating how ERP frameworks technologies have evolved to accompany new demands for flexible solutions would help understanding the technological hindrances to better ERP development. By doing so, it would be possible to identify process and product related problems that have been making ERP development sometimes become a failure.

References

- Alleman, G. B. Agile Project Management Methods for ERP: How to Apply Agile Processes to Complex COTS Projects and Live to Tell About It. In *Extreme Programming and Agile Methods: XP/Agile Universe 2002*, Springer Verlag, LNCS 2418, pp. 70–88.
- Ambler, S. *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*. Wiley, 2002.
- Ambler, S. When IT Gets Cultural: Data Management and Agile Development, *IEEE IT Professional*, vol. 10, no. 6, pp. 11-14, 2008.
- Bain, S. *Emergent Design: The Evolutionary Nature of Professional Software Development*. Addison-Wesley, 2008.
- Beck, K., *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 1999.

- Beck, K.; Andres, C., *Extreme Programming Explained: Embrace Change*. 2nd ed. Addison-Wesley, 2004.
- Beck, K.; Fowler Martin, *Planning Extreme Programming*. Addison-Wesley, 2000.
- Beck, K. et al. *Manifesto for Agile Software Development*. 2001. <http://agilemanifesto.org>
- Boehm, B. *Software Engineering Economics*. Prentice-Hall, 1981.
- Duvall, P. et al., *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley, 2007.
- Evans, E., *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2004.
- Fowler, M., *The New Methodology*. 2005. <http://martinfowler.com/articles/newMethodology.html>
- Fowler, M. *Continuous Integration*. 2006. <http://martinfowler.com/articles/continuousIntegration.html>
- Gamma, E. et al. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.
- Grudin, J *Interactive Systems: Bridging the Gaps Between Developers and Users*. Computer 59-69 ,1991.
- Jeffries, R. et al. *Extreme Programming Installed*. Addison-Wesley, 2000.
- Johansson, B. and Carvalho, R.A., *Management of Requirements in ERP development: A Comparison between Proprietary and Open Source ERP in XXIV ACM Symposium on Applied Computing*, 2009, Honolulu, p. 1605-1609.
- Kniberg, H., *Scrum and XP from the Trenches: How We Do Scrum*. 2007. <http://www.crisp.se/henrik.kniberg/ScrumAndXpFromTheTrenches.pdf>
- Koskela, L., *Test-Driven: TDD and Acceptance TDD for Java Developers*. Manning, 2007.
- Martin, R. *Iterative and Incremental Development*. 1999. <http://www.objectmentor.com/resources/articles/IIDII.pdf>
- Meszaros, G.; Aston, J. *Agile ERP: "You don't know what you've got 'till it's gone!"* Proceedings of the Agile 2007 Conference, Washington D.C., 2007, 143-149
- Poppendieck, M.; Poppendieck, T., *Lean Software Development: An Agile Toolkit*. Addison-Wesley, 2003.
- Raman, S., *Lean Software Development: Is It Feasible?*, in 17th Digital Avionics Systems Conference. 1998, IEEE: New York. 13-18.
- Schwaber, K., *Agile Project Management with Scrum*. Microsoft Press, 2004.
- Shen, H., Wall, B., Zaremba, M., Chen, Y., Browne, J. “Integration of Business Modeling Methods for Enterprise Information System Analysis and User Requirements Gathering”, *Computers in Industry*, vol. .54, n. 3, pp. 307-323, 2004.
- Shore, J. *Continuous Integration is an Attitude, Not a Tool*. 2005. <http://jamesshore.com/Blog/Continuous-Integration-is-an-Attitude.html>
- Sleeper, S.Z. *AMR analysts discuss role-based ERP interfaces - the user-friendly enterprise*. 2004; Available from: http://www.sapdesignguild.org/editions/edition8/print_amr.asp.
- Soh, C., Kien, S.S., and Tay-Yap, J., *Cultural fits and misfits: Is ERP a universal solution?* *Communications of the ACM*, 43,4 (2000), 47-51.
- Sutton, J.M., *Lean Software for the Lean Aircraft*, in 15th Digital Avionics Systems Conference. 1996, IEEE: New York. p. 49-54.
- Vernadat, F. B. “Enterprise Modeling and Integration (EMI): Current Status and Research Perspectives,” *Annual Reviews in Control*, vol. 26, p. 15-25, 2002.