

## **Agile Software Development for Customizing ERPs**

**Prof. Rogério Atem de Carvalho, D. Sc.**

**Fluminense Federal Institute, Brazil**

**Prof. Björn Johansson, Ph. D.**

**Copenhagen Business School, Denmark**

**Rodrigo Soares Manhães, M.Sc.**

**State University of North Fluminense, Brazil**

### **Abstract**

Customization of ERP systems is a complex task, and great part of this complexity is directly related to requirements management. In this context, a well-known problem is the misfit between the ERP functionalities and the business requirements. This problem comprises communication bottlenecks and difficulties on responding to changes. The proposals for minimizing these misfits are mostly focused on traditional, heavyweight waterfall-like approaches for software development. 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 chapter maps some of the main agile practices to ERP customization processes, using, where applicable, practices from a real-world ERP project. Moreover, some limitations on the agile approach to ERP customization are presented and discussed.

### **1. Introduction**

The last decades have witnessed two movements in industrial companies, namely Enterprise Resource Planning systems (ERPs) and Lean Manufacturing techniques. Lean techniques first appeared in the automobile industry within the so-called Toyota Production System, and afterwards were adapted and extended for application in practically all industry segments, forming the basis of what is now called Lean Manufacturing. Lean techniques are based on the principle that activities not delivering value to the final product should be eliminated from the production process. The lean techniques also, instead of just trying to anticipate demand fluctuations, try to follow fluctuations, by the use of the “pull” effect, where instantaneous demand is propagated from clients to suppliers, not the opposite, as was the usual approach some decades ago.

The term ERP was coined in the early 1990s by the Gartner Group (Wylie, 1990), and gained momentum in the years before the Y2K, considered the single event that signaled both the maturing of the ERP industry and the consolidation of large and small vendors (Jacobs and Weston, 2007). Blackstone and Cox (2005) define ERP as a “framework for organizing, defining, and standardizing the business processes necessary to effectively plan and control an organization so the organization can use its internal knowledge to seek external advantage.” ERP systems aim to realize integrated management, through the integration of business processes and the data they manipulate (Wier et al., 2007).

As was to be expected, some ERP vendors support Lean production planning and control techniques too, although some problems of this combination can be considered as still opened – since according to lean philosophy, some ERP features are too heavy (Seradex, 2009).

Development of ERP systems is an endeavor with a high level of complexity, and a great deal of this complexity is directly related to requirements management – being the misfit between ERP

functionality and business requirements a “classical” problem (So, Kien and Tay-Yap, 2000). The problem of misfit means that there is a gap between functionality offered by the package and functionality required from the adopting organization. Askenäs and Westelius (2000) describe this in the following way: “Many people feel that the current ERP system has taken (or been given) a role that hinders or does not support the business processes to the extent desire”. Another way of describing this is as said by Bill Swanton, vice president at AMR research, that only 35 per cent of the organizations are satisfied with the ERP system they use at the moment, and the reason for this dissatisfaction is that the software does not map well with the business goals (Sleeper, 2004). It can be argued that the misfit results from deficiency in the requirement management process, in which business analysts and developers are supposed to agree on what functionality the ERP system should support (Johansson and Carvalho, 2009).

Johansson and Carvalho (2009) suggest that the misfit problem is related to the requirements management process used in ERP development. In fact, the communication between business people and the development teams to a high extent occurs in a waterfall-based software process, 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’s necessary to define a customization (and maintenance) process that simultaneously enhances communication between ERP stakeholders and copes with change. 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 misfit or 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 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”, analyzing a real project in which 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<sup>1</sup>.

These methods are well accepted in the general software development community, however, there is no evidence that they are in use in the ERP development community. Additionally, Shehab et al. (2004), Esteves and Bohorquez (2007), and Botta-Genoulaz et al. (2005), show that there exists great extent of ERP research, however the major part is on deployment of ERP systems, being processes for customization a very poorly approached subject.

Therefore, the aim of this chapter is to propose a mapping of agile practices to ERP customization processes, using, where applicable, real world practices from the ERP5 project (Smets-

---

<sup>1</sup> 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.

Solanes and Carvalho, 2003). The authors believe that since Lean Manufacturing has proved itself as a way of providing a better production system to enterprises, there is an indication that Lean principles applied to software development can provide better ERP customization, by reducing the misfit problem and thereby increasing synchronization between business processes and process support in the ERP system, while reducing the waste of resources, such as time and money, in ERP customization. It is important to note that customization goes from parameterization to code change (adaptation or creation of new modules), being this second situation – that involves coding, is the one on which this chapter focuses. It is important to note that this work takes as a premise that the decision to modify the ERP code was already taken by the adopting organization.

## **2. Agile Software Development**

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. Most of these ideas are not new, but the innovation from Agile Manifesto's proponents was to add all this into a cohesive whole.

Fowler (2005) lists two fundamental differences on agile and traditional approaches to software development. First, agile approaches are adaptive rather than predictive. While predictive approaches 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 approaches 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 approaches, 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 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.

Brooks (1995) said it is impossible for customers to specify, in a complete and precise way, exact requirements of a relatively complex software project before effectively use some versions of that software. This means that working software is a valuable tool for requirements elicitation, what goes against the phased lifecycle of traditional, waterfall-based approaches, in which programming is a later activity. For achieving an effective learning, the software delivered should be a production-quality, full-featured release rather than a prototype.

Agile methods solve this question with the use of short iterations. 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 could 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 deep within the iteration which the feature is allocated.

During a project, the feature list can be changed several times reflecting its learning about the software being created and about the needs of the software. Agile methods is therefore a powerful tool for minimizing waste and promoting simplicity in the software development process, because decisions of what will be the scope is not made once in the beginning of the project, but iteratively in the project. Thus, only really needed features are implemented.

Currently, the most widely adopted agile-based 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 sections the main agile-related practices are discussed. It is important to note that some of them are directly related to agile practices, while others where embraced by these methods. Since the distinction of techniques' origins is not in the scope of this chapter, they are treated equally in this text.

### *Ubiquitous Language*

The concept of *ubiquitous language* is very important for communication, feedback and close collaboration between customers and developers (Evans, 2004). When communication between customer and the development team uses less documents and textual specifications and goes more to 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 for the whole project, covering the entire communication chain from customer and business analysis to the team's internal conversations and coding. The ubiquitous language reflects and feedbacks the knowledge 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. An

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”.

### *Test-Driven Development*

Test-Driven Development (TDD) is a technique that consists of writing test cases for any programming (new or feature adaptation, improvements, bug corrections etc.), before these implementations are performed. TDD is also known as *test-first programming*, which is the label when TDD is included as XP practices. Koskela (2007) said TDD is intended for “solving the right problem right”, meaning to achieve the correct solution that exactly matches the business problem.

For obtaining the correct solution, TDD prescribes automated unit tests, pieces of code that excite the code that has to be implemented. Thus, automated unit tests written before implementation lead the programmer to think and work in terms of contracts and collaborations between objects.

For matching business problems, TDD brings automated acceptance tests, which see the system as a black box. Thus, a full suite of acceptance tests fulfills, for most projects, any need for requirements documentation, having the advantage to be verifiable at any time. Furthermore, the collaboration between customers and developers provided by TDD is a tool for the improvement of the ubiquitous language (Crispin, 2006).

There are also integration tests, that tests collaboration of the units with the outside world (network, persistence, operating system and so on) and the collaboration of the units between themselves. There are several techniques and tools for supporting both unit, integration and acceptance tests, available for most programming languages and platforms.

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 fact, in an agile environment, up-front design is considered an anti-pattern (Ambler, 2002). In the TDD approach, the current design is only suitable in the current iteration, being the design improved for accommodating new requirements in every iteration. With good programming and design techniques, the design of a system can be continuously improved without to fall into the famous Boehm's cost of change curve (Boehm, 1981), which established that the cost of change in a software project increases exponentially through the time. 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.

For supporting this kind of evolving design, the unit tests are written in very small increments called *baby steps*. Baby steps, when applied to TDD, minimizes the time between an error and its discovering and keeps the complexity low (Beck, 1999). In this context, each baby step could be seen as one iteration in the TDD lifecycle.

### *Continuous Integration*

In several projects, integration brings up a bunch 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 software development 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). A CI environment must also have a single source code repository, usually managed by a version control system. Besides the source code, everything that is not automatically generated should be in the repository.

CI plays an important role on providing information just-in-time to projects. A CI environment provides a means to anyone get the latest executable and run it on his machine. Furthermore, most CI environments have a website, automatically updated by the build process, which gives useful information such as hosting status about the builds, test coverage rates and code inspection reports.

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 servers immediately warn the developers if something goes wrong, usually by e-mail, but even by SMS messages, visual warnings in light globes or big monitors, sound flags and so on, depending on project specificities.

### *Emergent Design*

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

Bain (2008) defines emergent design 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”. For a safe application of this principle, some technical disciplines must be applied, as TDD, refactoring, expressive code and a heavy use of patterns.

### *Expressive Code*

Applications are intended to be maintained for a long time, so it is safe to conclude that a program, in its whole lifetime, will be much more read than written, making code expressiveness a must. The key point to analyze code expressivity is that simple, clean and readable code matters (Beck, 2007). In fact, if the codebase is not good, programmers will face a number of problems: excessive

code complexity, bad design decisions obfuscated by confusing codification, excess of comments due to unreadable code - polluting the source file, difficulties on applying improvements or corrections and, consequently, rapid degeneration towards legacy code.

Accepting that source code is the ultimate software design (Reeves, 1992) and its tendency is to be as complex as the domain in which it is applied, makes code expressivity a quest to minimize the complexity of source code. Therefore code expressivity means choosing the simplest effective solution; avoiding over-engineering by using iterative and incremental approaches; applying both meaningful, plain English, pronounceable, non-abbreviated, domain-related, ubiquitous language-compliant and unambiguous names for all identifiers; avoiding any kind of duplication, applying optimization only where demonstrably necessary and so on. Expressive code is the code that reveals the programmer's intention with absolute clarity. Excellent fonts on coding best-practices and how to write expressive code are Beck (2007) and Martin (2008).

## 2.1 Agile and ERP

Others authors have already proposed the use of Agile methods in ERP customization. One of the first was Alleman (2002), who states that managing an ERP project is not the same of managing a large IT project. The ERP environment faces constant changes and reassessment of organizational processes and technology, therefore the project management method used with ERP deployment must provide adaptability and agility to support this evolutionary environment. Alleman (op. cit.) 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 (op. cit.) alerts that “the use of predictive strategies in this [ERP deployment] 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.” Alleman (op. cit.) goes forward proposing a different decision method in ERP project management, using an options approach to decision making, given that ERP projects has the flexibility to make changes to investments when new information is obtained, by treating this flexibility as an option allows decisions to be made in presence of uncertainty.

Meszaros and Aston (2007) present the specific case of the customization of a SAP/R3 module using agile principles and techniques. In this case study, major cultural and technical problems were identified:

- It is most accepted in the SAP community [and in every ERP community in general] that most companies should change their business processes to match SAP's. This “product works this way” goes against “whatever the customer wants” mindset of the agile philosophy.
- Developers' roles in SAP are quite separated, for instance ABAP programmers (who actually write the code) need to be helped by “Basis” specialists to install software. As well as “Functional” analysts are needed to configure components. This specialization of roles, besides going against the agile principle that a developer must be a generalist, also difficult effort estimation.
- Traditional SAP development is a document driven process, while agile philosophy emphasizes on person-to-person collaboration.

- ABAP development in SAP is server based. Without the capacity of creating individual development environments for running unit tests, a strategy of pessimist locks in code had to be assumed, creating a serious impediment for Continuous Integration.
- Frequent deployment, even in the ERP development environment, needed to go through strict governance processes, which forced discussions to identify the integration points and “pre test” the integration. Moreover, it uses typical waterfall Quality Assurance processes, focused on “could this possibly work” way of thinking, instead of the “does this line up with how the company wants to do things” agile way.
- There was the necessity of aligning the customization schedule to the release of SAP’s service packs, so that important new features wouldn’t be overlapped.

Seeking to overcome the cultural problems the development team, composed by programmers and the other necessary SAP’ s roles, was stimulated to work in a closer fashion, for instance sitting side-by-side to implement both coding and configuration. Regarding the technical problems, an “Agile Zone”, consisted of one dedicated developer workspace for each pair of developers, was created to address the server-based development issues. Although these measures had the expected effect, the costly SAP specialized personnel altogether with less-than-state-of-the-art development tools caused the final project cost hit more than the double of the estimated allocated costs for off-the-shelf tools and generalist professionals.

This chapter aims to contribute to this discussion by providing a mapping between the specific techniques used by Agile methods and ERP customization, being these techniques necessary to make the agile practices work. This proposal complements 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 an ERP using agile principles.

The next topic will discuss the mapping of the Agile practices described previously to ERP customization, focusing on programming and integration techniques, since the “higher” level techniques and philosophy were already described by other authors.

### **3. Mapping Agile and LD to ERP Customization**

When mapping Agile and Lean Development to the efforts of customizing an ERP system, it is important to clearly state what customization means. In the paper we define customization as the changes that are made to the system that are deployed in the using organization. This means that there is a need to distinguish between different ways of deployment. The basic difference needed to distinguish between is if the using organizations buy the software which they then host by themselves or if they buy the hosting of the ERP as a service. The difference between these is that changes made in the software as customization is done at different places and by different actors. In the external hosting case it can be said that customization is done closer to the vendor and that means that they have a “better” chance of learning from the changes but also to more easily implement the changes for other users. However, independent on who does the customization, customization means to a high extent to adjust the specific software to the business processes it should support. The basic thinking about customization is that it should be done for one specific instance of the software, which means that for the development of commercial off the shelf (COTS) products, such as ERP systems, could be seen as a paradox since the basic assumption is that the software should be so general that it could be sold as a

product attracting a broad customer base. However, customization exists and in some cases in a high extent.

Another question is if the kind of business model 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 ERP vendor sales directly to its customer) or indirect sale ERP (the vendor use a partner channel as distributor of the ERP), can influence how customization is done and, therefore, on how Agile practices can be applied. An answer to that question is that, as a general rule, the influence is more related to the customization level than to the business model. If customization is limited to parameterization then Agile techniques do not apply, while if customization goes beyond parameter adjustments there are code changes, Agile techniques can be applied; being the in-house development the extreme case of customization, where all Agile techniques can be applied.

### **Iterative and Incremental Development**

Customizing 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 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 customizing 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 if in later iterations, when requirements are better understood, it is realized that this process needs to be integrated to other processes, leading to changes in code and consequent rework?

This situation is here labeled as the “Late Integration Problem”, and will occur every time that the whole Business Process Modeling is not done before the actual customization 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 framework must provide fast and flexible process integration – preferably through configuration instead of coding, so that integration points identified lately during the requirements detail will not demand heavy rework. In that way it is possible to rapidly reconfigure and integrate business processes so that both incremental requirements elicitation and fast changing is achieved. 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 lately integration points and rapidly provide integration among the process.

To exemplify how this can be done in practice, ERP5's framework is briefly presented in the Annex. ERP5 shows how proper programming techniques can make a system highly flexible,

helping the adoption of Agile principles and thereby treating changes in the software as natural – and low cost - events.

### **Ubiquitous Language**

Creating a ubiquitous language is, as described earlier, 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. To deal with this problem, proprietary ERP vendors use different methods. Microsoft, for instance, in the development of their Dynamics products uses “Personas”. The basic thinking about Personas is that they are fictive users that are developed from interviews with a small sub-set of potential users, used to provide the developer with a better view over for whom they develop a specific system. To be able to do so the Personas are rich descriptions which should give the developer an understanding on who the users are, what they do and the context in which they do their work. However, even if Personas is one substitute for having a direct communication with the users it could be questioned if that is enough to higher customization levels. In those cases, the direct relation with real users is more efficient.

### **Test-Driven Development**

TDD is directly applicable to ERP customization, in special when dealing with high-level (black-box) testing. Given that an ERP will either substitute an exiting 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 an 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 assure 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 a incremental way, so when the system becomes more complex, testing also becomes more complex, but if the tests follow this complexity growing (as they are designed even before the actual coding), 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. Although this can be seen as an extra customization job, thorough testing is necessary for obtaining quality software and it is an important benefit. If all testing is automated, it is possible, even before actually implementing a customization (initial or

upgrade related) to test if the customization will work, by the use of mocks<sup>2</sup>. Moreover, by verifying side effects and errors identified during the tests using mocks, it is possible to enhance estimates for re-implementing customizations in the newly upgraded module.

Therefore, automated testing and mocking can reduce substantially the customization effort, even helping detecting integration problems between standard code and customized code before the second is totally implemented or re-inserted into upgraded modules.

## Continuous Integration

This technique can bring a series of benefits for ERP customization involving code changes. First, CI can help on continuously provide working versions of a customized module, which can be checked by users as modifications and new features are introduced, in a incremental way, reducing the risks of occurring misfits. Of course, 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.

Requirement: “For a given order, apply to it a trade condition negotiated between buyer and seller”.

Test Class: TestApplyTradeCondition()

Business Class Method: Order\_applyTradeCondition()

Figure 1: Simple example of expressive coding in ERP5<sup>3</sup>: a requirement generates a test and a method, all keeping very close, directly related names.

## Emergent Design

When it comes to ED and ERP customization it can be stated that this technique is limited, given that in any ERP environment<sup>4</sup> in general, ED has limitations in refactoring deepness,

<sup>2</sup> Mocks are objects that are injected into the unit under test and act in place of the real objects for testing purposes.

<sup>3</sup> The complete code can be found at <http://svn.erp5.org/erp5/trunk/products/ERP5/tests/testTradeCondition.py> and [http://svn.erp5.org/erp5/trunk/bt5/erp5\\_trade/SkinTemplateItem/portal\\_skins/erp5\\_trade/Order\\_applyTradeCondition.xml?revision=24616&view=markup](http://svn.erp5.org/erp5/trunk/bt5/erp5_trade/SkinTemplateItem/portal_skins/erp5_trade/Order_applyTradeCondition.xml?revision=24616&view=markup)

<sup>4</sup> In fact in any framework based environment.

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 we have the use of ABAP (Advanced Business Application Programming) 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.

#### **4. Limitations to the Agile Approach in ERP**

A series of limitations to the use of Agile methods in the current ERP realm can be identified, some of them, mainly the cultural ones, also happens with other types of systems, other 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 to 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 difficult of using Agile techniques such as Refactoring and Emergent Design (Ambler, 2008)<sup>5</sup>.

##### **Language Limitations**

- 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,

---

<sup>5</sup> In fact that's a half cultural and half technical limitation.

however, does not eliminate the duplicity of languages (non-compliant to UL in the ERP itself and compliant in the customization).

## 5. Conclusions and Research Directions

This chapter starts on the premise that the adopter have 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 the development and in that way it could be a change towards agile methods (and their Lean principles). Moreover, the statistics on unsuccessful ERP implementations urges for new ways of facing the customization problem.

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 development 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 to deliver the software.

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 development approaches are adaptive rather than predictive. This means that if it is possible to implement them in ERP customization, 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 software development for customizing ERPs 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 the Agile and Lean practices adoption level in ERP customization, differentiating the proprietary, free/open source, and in-house solutions. Also, investigating how ERP frameworks technologies have evolved to accompany the new demands for flexible solutions would help understanding the technological hindrances to better ERP deployment. In that way, it would be possible to identify the process and product related problems that have been making ERP deployment sometimes 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.

Askenäs, L. and Westelius, A. *Five roles of an information system: a social constructionist approach to analyzing the use of ERP systems*. in *twenty first international conference on Information systems*. 2000. Brisbane, Queensland, Australia: Association for Information Systems.

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*. 2<sup>nd</sup> ed. Addison-Wesley, 2004.

Beck, K. *Implementation Patterns*. Addison-Wesley, 2007.

Beck, K.; Fowler Martin, *Planning Extreme Programming*. Addison-Wesley, 2000.

Beck, K. et al. *Manifesto for Agile Software Development*. 2001. <http://agilemanifesto.org>

Blackstone Jr., J.H., Cox, J.F., 2005, p. 38. APICS Dictionary, 11th ed. APICS: The Association for Operations Management.

Boehm, B. *Software Engineering Economics*. Prentice-Hall, 1981.

Botta-Genoulaz, V., Millet, P.A., and Grabot, B., A survey on the recent research literature on ERP systems. *Computers in Industry*, **56**,6 (2005), 510-522.

Brooks, F. P., *The Mythical Man-Month*. Addison-Wesley, 1995.

Bulka, A. (2001, August). Relationship Manager Pattern [Online]. Available: <http://www.atug.com/andypatterns/rm.htm>

Carvalho, R. A.; Campos, R. ; Monnerat, R. M. . ERP System Implementation from the Ground up: The ERP5 Development Process and Tools. In: Muthu Ramachandran, Rogerio Atem de Carvalho. (Org.). *Handbook of Research on Software Engineering and Productivity Technologies: Implications of Globalisation*. IGI Global, 2009, p. -.

Carvalho, R. A.; Monnerat, R. M. . Development Support Tools for Enterprise Resource Planning. *IT Professional*, v. 10, p. 39-45, 2008.

Carvalho, R. A.; Monnerat, R. M. . ERP5: Designing for Maximum Adaptability. In: Gregory Wilson, Andy Oram. (Org.). *Beautiful Code: Leading Programmers Explain How They Think*. Sebastopol, EUA: O'Reilly Media, 2007, p. 339-351.

Crispin, L. *Driving Software Quality: How Test-Driven Impacts Software Quality*. In: *IEEE Software*, vol. 23, n° 6, pp. 70-71, 2006.

Duvall, P. et al., *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley, 2007.

Esteves, J. and Bohorquez, V., An updated ERP systems annotated bibliography: 2001-2005. *Communications of AIS*, **2007**,19 (2007), 386-446.

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>

Fowler, M. *AgileVersusLean*. 2008. <http://martinfowler.com/bliki/AgileVersusLean.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.

- Jacobs, F. R., Weston, F.C., Enterprise resource planning (ERP) —A brief history, *Journal of Operations Management*, Volume 25, Issue 2, March 2007, Pages 357-363.
- 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 *SAC conference*. 2009 Forthcoming: Hawaii.
- Johnson, J. *ROI, It's Your Job*. Published Keynote on the 3<sup>rd</sup> International Conference on Extreme Programming, Alghero, Italy, 2002.
- 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>
- Martin, R. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice-Hall, 2008.
- 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
- Monnerat, R. M.; Carvalho, R. A. ; Campos, R. . Enterprise Systems Modeling: the ERP5 Development Process. In: 23rd Annual ACM Symposium on Applied Computing, 2008, Fortaleza. Proceedings of the Proceedings of XXIII ACM Simposium on Applied Computing. New York : ACM, 2008. v. II. p. 1062-1068.
- 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.
- Reeves, J. *What Is Software Design?* C++ Journal, Fall 1992. Available from: [http://www.developerdotstar.com/mag/articles/reeves\\_design.html](http://www.developerdotstar.com/mag/articles/reeves_design.html)
- Schwaber, K., *Agile Project Management with Scrum*. Microsoft Press, 2004.
- Seradex. *Lean Manufacturing - Seradex ERP Solutions*. 2009 [cited 2009 February 5]; Available from: [http://www.seradex.com/ERP/Lean\\_Manufacturing\\_ERP.php](http://www.seradex.com/ERP/Lean_Manufacturing_ERP.php)
- Shehab, E.M., Sharp, M.W., Supramaniam, L., and Spedding, T.A., Enterprise resource planning: An integrative review. *Business Process Management Journal*, **10,4** (2004), 359-386.
- 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](http://www.sapdesignguild.org/editions/edition8/print_amr.asp).
- Smets-Solanes, J. ; Carvalho, R. A. An Abstract Model for An Open Source ERP System: The ERP5 Proposal. In: VIII International Conference on Industrial Engineering and Operations Management, 2002, Curitiba, Brazil.
- Smets-Solanes, J-P. ; Carvalho, R. A. . ERP5: A Next-Generation, Open-Source ERP Architecture. *IEEE IT Professional*, v. 5, n. 4, p. 38-44, 2003.

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. 49-54.

Vernadat, F. B. "Enterprise Modeling and Integration (EMI): Current Status and Research Perspectives," *Annual Reviews in Control*, vol. 26, pp. 15-25, 2002.

Wier, B., Hunton, J. and Hassab-Elnaby, H. R. (2007) Enterprise resource planning systems and non-financial performance incentives: The joint impact on corporate performance. *International Journal of Accounting Information Systems*, 8, 165-190.

Wylie, L., 1990. A vision of the next-generation MRP II. Scenario, 300-339, Gartner Group, April 12, 1990.

## Key Terms

### ERP

Enterprise Resource Planning system, a kind of software which main goal is to integrate all data and processes of an organization into a unified system.

### ERP Customization

In the paper ERP customization is defined as code changes that are made either with the aim of adaptation or creation of new modules, which means that ERP customization from the adopters point of view already are decided upon beforehand. ERP customization is thereby not something that is done on already deployed code.

### ERP Customization Life Cycle

The set of activities accomplished to customize and keep the customization code up-to-date with new system upgrades and new requirements, during the whole ERP particular instance lifecycle.

### Late Integration Problem

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. When in later iterations, as requirements are better understood, it is realized that this process needs to be integrated to other ones, changes in the code will be necessary, incurring in rework. This situation will always occur when the entire Business Modeling is not done before the actual development or customization of the system.

### Business Synchronization

Synchronization means that the organization's business processes and the supporting technology evolves in tandem so that when either of these parts changes the other part adjust to the change. Business synchronization then means that there exists organization-technology synchronization

### Agile methods

Agile methods are a generic label for development and/or project management methodologies that view software development as a learning process, and prescribe close collaboration with the customers; working, production-quality software released at short iterations; code-centric development; fast feedback chain; simplicity; and the elimination of processes that do not aggregate value to the product. Agile methods emphasize people rather than processes and have delivering business value as the ultimate measure of progress.

### Lean Development

Lean Development is a mapping of Lean concepts to the software development domain. It is based on Lean principles as waste elimination, team empowerment, built-in quality, late irreversible decisions and so on. In recent years, Lean Development has gained momentum within the Agile community, since many of its propositions match agile views.

## Annex – ERP5 Basic Principles

This annex briefly presents the ERP5 framework, showing how it can support the use of Agile and Lean Development techniques in ERP customization. A Model Driven approach for developing enterprise systems on top of ERP5 can be found at Carvalho, Campos and Monnerat (2009).

### ERP5

The ERP5 project (Smets-Solanes, Carvalho, 2002; Smets-Solanes, Carvalho, 2003) is a Free and Open Source (FOS-ERP) that aims at offering an integrated management solution based on the open source Zope platform, written in the Python scripting language. This platform delivers an object database (ZODB), a workflow engine (DCWorkflow), and rapid GUI scripting based on XML. Additionally, ERP5 incorporates data synchronization among different object databases and an object-relational mapping scheme that stores indexing attributes of each object in a relational database, allowing much faster object search and retrieval, in comparison to ZODB, and also analytical processing and reporting. This project was initiated in 2001 by two French companies, Nexedi – its main developer, and Coramy – its first user, and since then is in development and use by a growing community from France, Brazil, Germany, Poland, Senegal, Japan, and India, among others. ERP5 is named after the five core business entities that define its Unified Business Model (UBM, Figure 1):

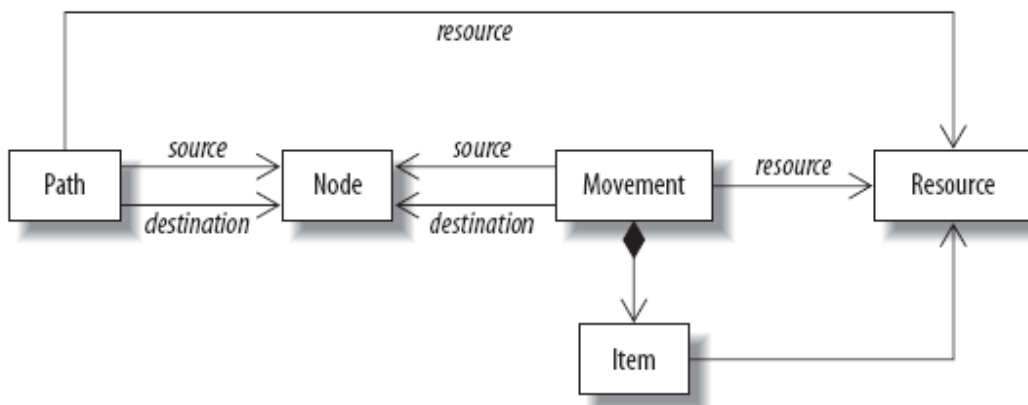
*Resource*: describes an abstract resource in a given business process (such as individual skills, products, machines etc). Material lists, as well as prototypes are defined by the relationship between nodes.

*Node*: a business entity that receives and sends resources. They can be related to physical entities (such as industrial facilities) or abstract ones (such as a bank account). Metanodes are nodes containing other nodes, such as companies.

*Path*: describes how a node accesses needed resources.

*Movement*: describes a movement of resources among nodes, in a given moment and for a given period of time. For example, such movement can be the shipping of raw material from the warehouse to the factory.

*Item*: a physical instance of a resource.



**Figure 1. ERP5 Unified Business Model.**

The structure of an ERP5 instance is defined through mappings of the particular instance concepts to the five core concepts and supportive classes or, in very rare cases, through the extension of the UBM. This mapping is documented by a proper instance's lexicon. For example, debit and credit values can be mapped to the Quantity property of the Item class. Its behavior is implemented through workflows, which implement the business processes, and consider the concept of Causalities (chains of related events). Very flexible and extensible modules, called Business Templates, are also provided for Accounting, Production Planning, Payroll, Finance, MRP, CRM, Trading, Electronic Commerce, Reporting, and others. Additionally, a project management toolset is provided to support both the customization and development processes (Carvalho and Monnerat, 2008).

### Using ERP5 Framework to Support Agile Development

Among the various features in the ERP5 framework, that make it flexible, some of these can be directly mapped as supportive to an Agile development process, in special, three of its design patterns, providing rapid customization, which makes the framework very adaptive to changes in business requirements.

The prototype pattern is the most obvious design pattern in ERP5, a thoroughly explanation on its use within this framework can be found in Carvalho and Monnerat (2007). ERP5 Modules work as Clients; asking concrete prototypes to get instantiated (clone themselves). In that way, business processes can be implemented on top of the UBM with few programming, using

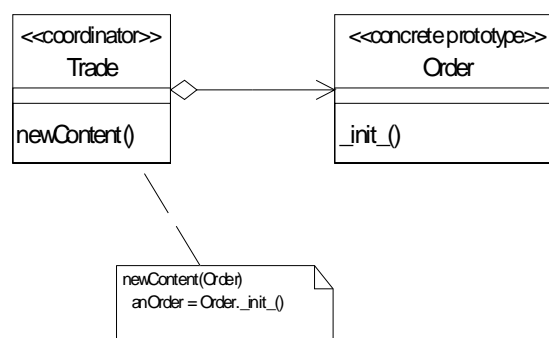


Fig. 2. Prototype pattern implementation. The Trade module calls the template method newContent, provided by ERP5 core to create new concrete prototype instances. The typical pythonic \_init\_() constructor is the equivalent to the Clone() operation of the pattern.

template structures and algorithms. Concrete prototypes provide masked names for abstract terms – for instance, attributes like “movement source” and “movement destination”, when used in an implementation of a money transfer transaction, are translated into “source account” and “destination account”. ERP5 core provides a template method called newContent, which creates concrete prototype objects according to a concrete class’ name parameter. Concrete subclasses use the GUI to mask elements in accordance to specific business terms, which are stored on a documentation tool called Lexicon, responsible for keeping track of these mappings (Carvalho and Monnerat, 2008).

The use of prototype objects makes it easy to reuse and compose UBM elements. However, it is still necessary to solve the Late Integration Problem. For achieving this, two other design patterns are deployed, namely Mediator (Gamma et al., 1996) and Relationship Manager (Bulka, 2001). In ERP5, mediators are called Builder objects, and are used to integrate the behavior of both UBM elements and system modules, or, in other words, mediators are responsible for inter and intra-module integration. These objects are Singletons that reduce coupling among business entities’ workflows, like Delivery and Order; and among modules’ workflows, such as Trade and Accounting. When integrating UBM elements behavior, they form macro-processes, which in turn describe the behavior of the modules. Figure 3 exemplifies how Builder objects are used.

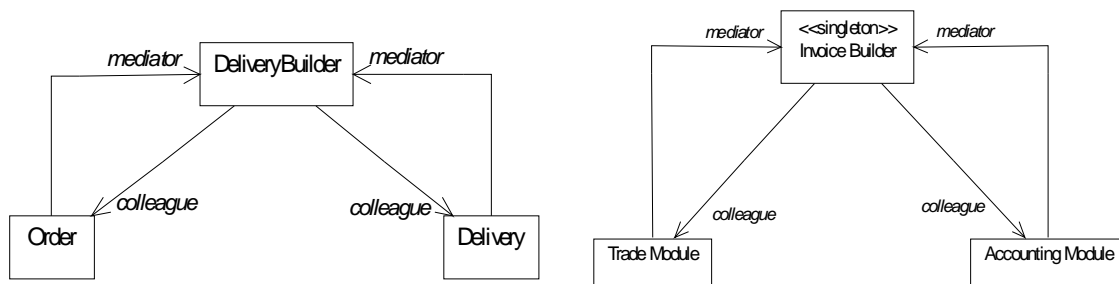


Fig. 3a. Mediator pattern for intra-module integration. The delivery builder object mediates the communication between order and delivery, reducing behavior coupling between both classes. For instance, in an implementation for a logistics company, after confirmed, an order object’s workflow signals to the builder that it can start the creation of a new delivery and start its workflow. In an implementation for a manufacturing company, an order would signal to a production order builder object, with no need to change the workflow configured for Order objects.  
 Fig.3b. Mediator pattern for inter-module integration. The invoice builder integrates trading and accounting. Even if the Accounting module is installed long after the Trade module, it is possible to generate invoices for all trading transactions executed in the past.

While Builder objects are used to integrate workflows by creating appropriate calling chains, it is still necessary to connect the objects that are used to represent the business entities that collaborate to realize these processes, providing navigation facilities among them. Coding relationship logic into each business class is often a tedious and error-prone task. Also, traditional relationship code means to spread out code (back-pointers, deletion notifications, etc) amongst many classes, which is more difficult to track, maintain and keep in synch than mediated approaches (Bulka, 2001). Therefore, while mediators are used to integrate behavior, Relationship Managers are used to integrate structure. Actually, in ERP5 a central relationship manager, called Portal Categories, is used to maintain all the one-to-one, one-to-many and many-to-many relationships between groups of related objects. Query methods and relationship code are provided. The Portal

Categories is a Singleton object that generates code for receivers and senders automatically, and contains Base Category objects, which in turn are responsible to connect classes that collaborate in a given business (macro) process. Figure 4 presents an example use of Portal Categories.

Given that both Builders and Base Categories objects provide all code necessary to integrate business entities, they are configurable structures, therefore when new business processes integration points are identified, it is just a question of configuring the proper relationships to provide integration. By providing ways of rapidly and cheaply integrating structure and behavior, ERP5 framework allows late business integration during the customization process, allowing an

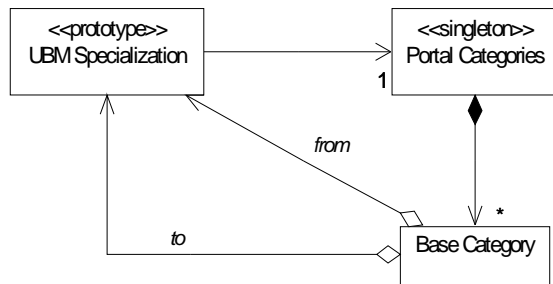


Fig. 4. Relationship Manager Pattern implementation. The Portal Categories object offers a complete relationship management service to objects. This service, using configurable rules, creates the necessary getters, setters, and references.

incremental process with its late discovery of integration points. In that way, it is possible to improve the sometimes dynamic and cross organizational requirements for management of collaborative processes.