Self-organizing Business Networks, SOA and Software Maintenance

Francesco Rago

***Abstract*—These Actual business leverages on software service to improve general firm performance. Software is always more a strategic asset to sustain business. In meantime business is changing continuously: there is a transition from centralized to distributed and cooperative organizations. This work describes a software maintenance process strictly integrated with enterprise evolution. It starts from the basic hypothesis that the software service model has to be integrated in the process of define/improve business models at run time and not in a separate step. This means paradigmatic changes: from top down maintenance and control to a bottom-up evolutionarily life cycle where software assets maintenance is integrated with organizational assets maintenance. The benefits of the model are measured reduction of the number of defects on high level requirements and the incremental commitment nature of the process: expenditures tend to be balanced with certainty level.**

***Index Terms*—Software process for maintenance, Services Organization Architecture (SOA).**

I. Introduction

Business models of firms are changing: there is an evolution toward distributed models better suited for integration into a global economy. This scenario is well described by Paul R. Krugman in [1] and implies a strategic capacity to manage new emerging values, strategies and products/markets. The enterprises organizational models have to consider a network of internal and external agents (partners) with fuzzy boundaries and continuous exceptions in processes. Such status is the source of new ideas and opportunities [2]. This environment is particularly challenging for software maintenance and new “smart” approaches have to be defined.

The shift from centralized to distributed and cooperative organizations needs software with Service Oriented Architecture dynamically integrated with business architecture. Software applications become the fundamental platform to support products delivery and services management. Their maintenance is becoming a major challenge to guarantee software aligned to business processes.

This work describes a maintenance process designed to satisfy the described scenario. The innovative aspect derives from an agile approach integrated with self-organizing and changing business. The self-organization process is not described in this report, because out of scope.

F. Rago., is with M3 Comp. LLC, 113 Barksdale Professional Center,Newark – DE (USA) (e-mail: francesco.rago@megatris.com).

II. . BUSINESS CONTEXTS DEFINITION

The context concept is a fundamental notion of an Enterprise Architecture [3]. Contexts are typically used to model the effect of the environment on interactions and communications occurring among active (and typically intelligent) agents, such as humans or artificial. Contexts typically describe interactions and communications.

III. AGENTS SYSTEMS

The multiple interacting agents in economy models can represent individuals (e.g. people), social groupings (e.g. firms), biological entities (e.g. growing crops), and/or physical systems (e.g. transport systems) [4]. In particular, system events should be driven by agent interactions. Agent systems are not control-oriented, but task-oriented. They have no central control authority, instead each agent is an independent locus of control. Control is inside agents and the agent’s task drives the control. They may be anything human or humans or artificial inside the value chain of the firm. Control and data are uncoupled, since data do not necessarily flow with control [5]. Agent coordination contexts can serve the purposes of:

- enabling agents to model the environment where they interact and communicate (the subjective viewpoint),
- providing a framework to express how the environment affects interpretation of agent communication acts (the objective viewpoint). The notion of agent coordination context enables agents to perceive the space where they act and interact, reason about the effect of their actions and communications, and possibly affect their environment to achieve their goals.

IV. SELF-ORGANIZATION CONTEXT OF FIRM AND ECONOMICS: THE ROLE OF SOFTWARE MAINTENANCE

Self-organization refers to the capacity of networks of agents have for combining and recombining capabilities without a centralized and detailed managerial guidance. Since technology is the engine that drives the most successful economies of post-industrial societies, it seems likely that technological innovation is characterized by self-organizing processes in nets of firms [6].

Networks of agents are linked organizations with no central control authority. They create, acquire and integrate diverse knowledge and skills required to manage complex technologies and products. An agents model is appropriate for networks of firms that are in a supply chain ruled by Service Level Agreements.

Network self-organization involves a continuous interplay among three sets of factors: network resources, constraining and focusing factors. Taken together, these factors comprise what might be called a “resource-based view of the network”. The network resources have at least existing core capabilities, internalized software assets and ability in organizational learning. Learning is the key of the process because new capabilities and software assets are identified, acquired, shared among network participants, and continuously updated or discarded to give to the network as a whole competitive leadership in a particular market sector. Learning through collaboration and effective self-organization requires that networks have a “window” on their partners’ capabilities and assets. Network learning is thus inseparable from the evolving interactions among holders of core capabilities and complementary software assets that are an ongoing part of self-organization network. This means a paradigmatic change: from top down maintenance and control to a bottom-up and evolutionarily method, integrating software assets maintenance with organizational assets. Software has a strict relationship with the other components of the enterprise architecture and each agent is integrated in the process of software maintenance.

V. Business Model for I.T. Self-Organization Processes

A Business Model describes the architecture of the firm’s business and its network of partners for creating, marketing and delivering value and relationship capital, in order to generate profitable and sustainable revenue streams. Networks of agents have to share all or part of the model to integrate their activities and software assets.

Adopting a definition from Hagel and Singer (1999) [10], a business model has to address the following topics (see fig. 1):

- [Agent] internal and external agents including Customers that operate for and against business goal,
- [Activity] the business, the product innovation and the value proposition offered on the market.
- [Relationship]&[Partnership] the customers targeted, how to deliver them the products, and the relationships with them,
- [Channel] Organizational structure based on agents network,
- [Infrastructure Network] the logistics infrastructure together with Hardware/Software assets,
- [Revenue]& [Costs] the revenue model and the cost model.

The Infrastructure Network contains the list of hardware and software assets. Maintenance of software assets operate on this part of the model and of its relationships.
A Business Model has to be shared with all interested agents. To maintain the model means also to be aware of business evolution and to be able to modify software in time.

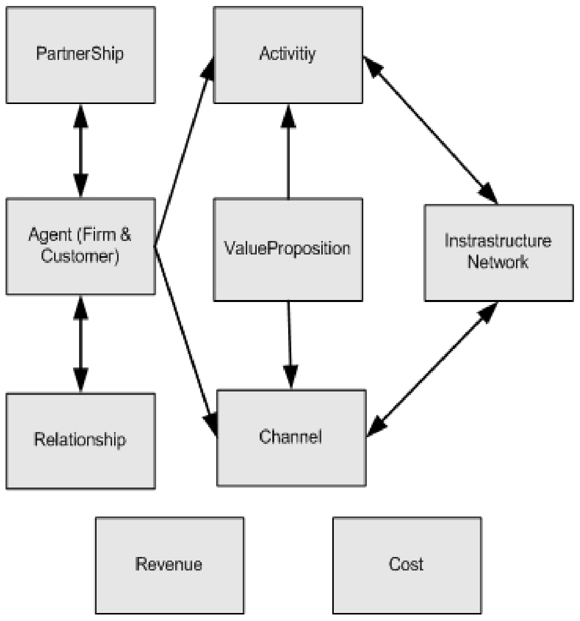

Fig. 1 Self-Organization Business Model Ontology

VI. Software Maintenance of Systems

Once a business model is defined and agreed by all agents, we used a method to support software maintenance through advanced iterative and incremental approach. We adopted a process similar to SCRUM [11]-[13] because it is not a step-by-step cookbook approach and requires active, thoughtful development and management. The method starts with the premise that maintenance environment is complicated and unpredictable.

You can predict or definitively plan what you will deliver, when you will deliver it, and what the quality and cost will be, but you have to negotiate continuously them according to various risks and needs as you proceed. SCRUM method was integrated with the Enterprise Business Model maintenance to improve the associated SOA model and the software architecture.

The fundamental steps of an extended SCRUM method follow:

Stage I. Concept

The purpose of the concept stage is to better define exactly how the business model and its context was changed/improved, who is aimed at, how it will be positioned in market segments and how Information Technology assets has to be changed/improved to support business focus on target.

The Business Model is analyzed with a process approach using Statistical Process Control results (where applicable) to value the actual status of processes.

The following main topics are relevant:

- social goals and effectiveness of agents commitments;
- possible interferences or agents opportunities;

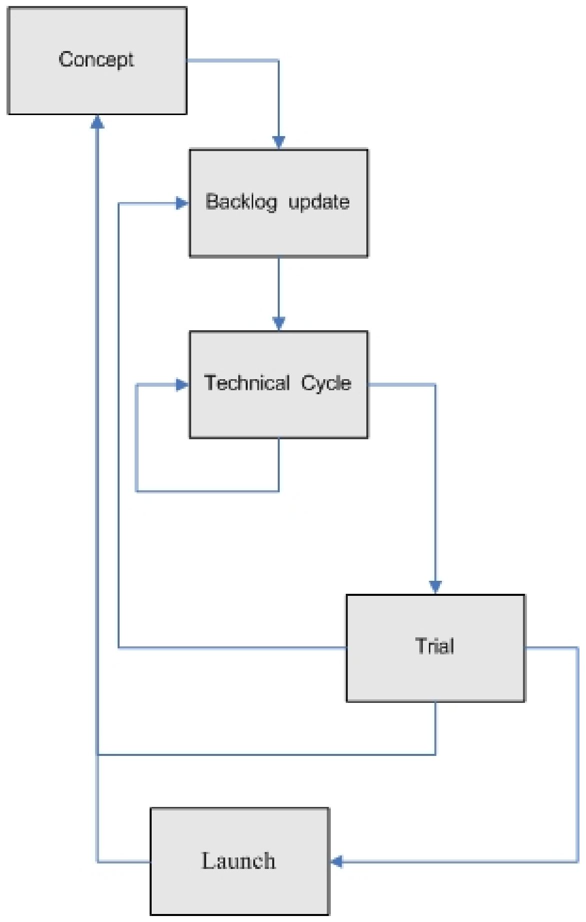

Fig.2 Complete Process

- links between agents (flow of information regarding other agents' epistemic utility to improve measurable performance [15]);
- presence of signs or exceptions relative to "little piece of business" able to reveal tendencies or new business evolutions. This means the identification of possible business evolutions, because not always a stable and capable process is a successful process for ever.

The main rules to model the architecture of Hardware and Software are on the basis of this approach:

- Each package-subsystem has to serve a set of primary functions/services to guarantee meaningful business coverage.
- The function/services have to guarantee stakeholders utility and have measurable statistical capabilities.
- Each package-subsystem have to be of limited dimension (size). This helps interactions and maintenance.
- Each package-subsystem has to manage strong data quantities with integrated sub functions to measure performance of the business functions.

Stage II. Backlog update

The software maintenance begins in earnest. It is used a streamlined, flexible approach to requirements change management reflecting both Extreme Programming (XP)'s planning game and the SCRUM methodology.

There is a stack of prioritized and estimated requirements which needs to be implemented. Such requirements are a consequence of the Stage I analysis and have the scope to guarantee agents successful operations. SCRUM suggests that the requirements are frozen for the current iteration to provide a level of stability for the developers.

The Product Backlog is generated on requirements and used during the entire development phase. The Product Backlog contains a prioritized list of all the activities, Since the Product Backlog is never finalized, the priority, along with the listed items, can be changed during the entire maintenance phase.

Stage III. Technical Cycle

This is an iterative stage where technical operations are accomplished. The structure of the service application is created/maintained. The stage has the following steps:

III.1 Create/Update Solution architecture

Define/Update the conceptual structure of the service application.

III.2 Define/Update runtime environments

Define the runtime environments in which the Service application should run. This covers all test environments, including unit test and final production environment.

III.3 Identify Existing Models and Common Patterns

The repeating patterns are identified within the service application. These patterns often occur either because of the consistent use of an architectural style or because of the requirements of the runtime platforms. Common patterns are compared with existing assets, making any necessary small adjustment to their architecture to exploit what is already available.

III.4 Define/Update design model

This task creates/updates the definition of a model that specifies the components for the Service in a runtime-independent manner.

III.5 Design, code, test

Step of design, code and test that produces the service software.

Stage IV. Trial

The Trial stage is a validation of the product's design and features in use. Software prototypes are tested within the firm to determine that no technical flaws exist. In parallel, an agent test of the product is conducted. The object is to identify design defects, and, in particular, modifications needed to improve business agents' acceptance. The trial stage represents a "dry run" of all commercial facets of the software. The agents' tests provide the inputs to finalize the business model if issues appear in the new enterprise architecture. This means the identification of needed adjustments to the business model. A final estimate of market share and expected sales are two results of the test market.

Stage V: Launch

The launch stage involves startup of full or commercial production and the implementation of infrastructure assets of the business model. Post launch evaluation or control points at pre-designated times after launch provide benchmarks to gauge whether a software product is "on target." Such benchmarks include market share, sales volume, production costs, etc. Post-launch evaluations are essential to control the software product and to signal the implementation of corrective schemes to move the software product back on course.

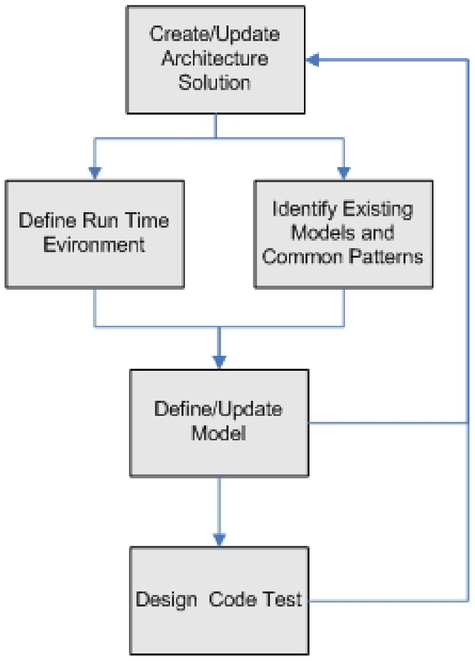
Create/Update Architecture Solution
Define Run Time Evironment
Identify Existing Models and Common Patterns
Define/Update Model
Design Code Test

Fig.4 Technical Cycle

VII. RESULTS

The presented approach was used in four closed web small and midsize projects with the following post mortem results.

People culture is still not aligned to service approach. Managers think in term of functional black box, with strictly defined boundary following a typical top-down engineering approach. This is a problem in the beginning of the innovation process. It is difficult to reason in term of knowledge and services sharing on common goals in a network. This has a heavy impact on the start-up of software process with an over-cost of 30-40% of effort.

The real improvement was measured in the number of defects on high level requirements: there was a meaningful reduction of 30-40%. The average number of defects for project was 12.4 [defects/N.of Projects] during Concepts phase while the previous average with different life cycles (Requirements and Analysis phases) was 18.7. This gave a positive answer to the basic hypothesis that the software service model has to be integrated in the process of define/improve business model at run time and not after in a separate step.

The complexity of the maintenance process grew because Product Backlog had to parallelize different and often conflictive changes. This happened in the 40% percent of the cases.

Main data of closed projects are contained in Table I.

Project	Technology	K-loc s	Changes	Concept Defects
1.	C/C++	79368	47	3
2.	Asp/ C++	2593	31	15
3.	ASP/ C++	2680	45	10
4.	Php/ C++	1680	5	21

Table 1: Changes vs Defect

VIII. CONCLUSION

New software product/service maintenance will never be risk free. Much can be learned about effective new software management from a review of the experiences in past projects. Many of these insights have been incorporated into the method presented. The benefits of the model are many. One result is that the process becomes more multidisciplinary. The balance between the internal versus external orientation becomes obvious. A second payoff is that interaction between agents is encouraged: many evaluation nodes demand diverse inputs from different groups in the company. A third benefit is the incremental commitment nature of the process: expenditures tend to be balanced with certainty level; each stage involves progressively better information and concurrently entails progressively higher expenditures; and risk is managed. Further, decision nodes and bail-out points are provided at each stage. Finally, the process is market oriented, providing for ample market information and marketing planning, not only towards the launch phase, but throughout the entire process.

REFERENCES

[1] Paul R. Krugman, The Self-Organizing Economy, Cambridge, MA: Blackwell Publishers, 1996; Jose A. Scheinkman and Michael.
[2] Peter A. Corning, “Synergy and Self-Organization in The Evolution of Complex Systems,” Systems Research, June 1995, pp. 89-121.
[3] Stafford Beer, Diagnosing the System for Organisations; 1985, John Wiley.
[4] Bruun, Charlotte (Ed.), Advances in Artificial Economy as a Complex Dynamic System, Series: Lecture Notes in Economics and Mathematical Systems , Vol. 584.
[5] Lucio Biggiero, “Self-Organizing Processes in Building Entrepreneurial Networks: A Theoretical and Empirical Investigation,” Human Systems Management, No. 3, 2001.
[6] Mary E. Lee, “The Evolution of Technology: A Model of Socio-Ecological Self-Organization,” in Loet Leydesdorff and Peter Van den Besselaar, eds., Evolutionary Economics and Chaos Theory: New Directions in Technology Studies, New York: St. Martin’s Press, 1994.

[7] Frank-Jurgen Richter, “The Emergence of Corporate Alliance Networks—Conversion to Self-Organization,” Human Systems Management, No. 1, 1994, pp. 19-26.
[8] Richard N. Osborn and John Hagedoorn, “The Institutionalization and Evolutionary Dynamics of Inter-organizational Alliances and Networks,” Academy of Management Journal, April 1997.
[9] Anders Lundgren, Technological Innovation and Network Evolution, New York: Routledge, 1995, pp. 77-104.
[10] Hagel, J., Singer, M. (1999) Unbundling the corporation. Harvard Business Review 77 (2): 133- 141.
[11] Linda Rising and Norman S. Janoff, The Scrum Software Development Process for Small Teams., AG Communication Systems.
[12] Ken Schvaber and Mike Beedle, Agile Software Development with Scrum, Prentice Hall. USA.
[13] Jim Highsmith, Alistair Cockburn, Agile Software Development: The Business of Innovation, September 2001.
[14] R. J. Bril , R. L. Krikhaar , A. Postma , Architectural support in industry: a reflection using C-POSH, Journal of Software Maintenance and Evolution: Research and Practice, Volume 17, Issue 1 , Pages 3 – 25.
[15] Wynn C. Stirling, Coordinated Intelligent Control Via Epistemic Utility Theory, Control Systems, Vol. 13 n.5.
[16] Scott M. Brown,Norman Wilde, A Software Maintenance Process Architecture, 9th Conference on Software Engineering Education (CSEE),1996.
[17] Keith H. Bennett, Václav T. Rajlich ,Software maintenance and evolution: a roadmap, Proceedings of the Conference on The Future of Software Engineering, 2000.