Reconfigurability Function Deployment in Software Development

Stelian BRAD, Adrian CHIOREANU, Mircea FULEA, Bogdan MOCAN, Emilia BRAD
Technical University of Cluj-Napoca, Romania
stelian.brad@staff.utcluj.ro, adrian.chioreanu@com.utcluj.ro
mircea.fulea@staff.utcluj.ro, bogdan.mocan@muri.utcluj.ro, emilia.brad@muri.utcluj.ro

In the forthcoming highly dynamic and complex business environment high-speed and cost-effective development of software applications for targeting a precise, unique and momentary set of requirements (no more-no less) associated to a customized business case will bring significant benefits both for producers and users. This requires a life cycle change-oriented approach in software development. In this respect, designing software with intrinsic evolutionary resources for reconfiguration represents the sound approach. A methodology for concurrent deployment of reconfigurability characteristics in software applications is introduced in this paper. Its potential is exemplified in a case study dealing with web-based software tools to support systematic product innovation projects.
Keywords: *Reconfigurability, Software Development, Innovation, TRIZ, RAD*

1 Introduction

Software development methodologies date back from the 1960s and provide a frame for structuring, planning and controlling the actual software development process [1]; in other words, they support the entire life cycle of a software product, describing what is to be performed in each stage of the development process [2] [3].
Software development methodologies are chosen at the be-ginning of a project. When deciding upon a specific methodology, the development team should consider the complexity of requirements, particularities of the design and coding team, complexity of the overall project, and how the software product should evolve.
As businesses and technologies changed, development teams should face not only complex software requirements, but dynamic ones. Scenarios might arise when requirements can't be even completely known when the actual coding phase starts. Factors which have increased the complexity of requirements arise from intensified global competition, reduction in lead-time and life expectancy of products, diversification of demand, and new technologies [4].
A popular software development methodology able to face such challenges is the agile concept, which deals with irregular and unpredictable requirements [5] [6]. It emerged from concepts as Rapid Application Development (RAD), prototyping and evolutionary (e.g. spiral) life-cycle models [7]. The basic goal of the agile software development is to combine the life-cycle approaches mentioned above with adaptive and collaborative team work practices in order to obtain short design cycles, which would finally reflect the requirement dynamics.
However, all existing methodologies (including agile) rely on the idea that requirements are eventually determined at the time the product reaches its beta or release-candidate stage. To deal with ambiguous requirements, different approaches must be taken into account.

2 The problem

According to some visionary analyses, the increasing dynamics of business environment will have significant implications on software development approaches, too [8]. In this respect, the static software engineering approach “plan-design-develop” will be replaced by the dynamic approach “anticipate-predict-learn” [9]. Among other issues, a key focus will be also on postponing the major decisions from the design-phase (as usually it is done in the traditional software life cycle) at the run-time phase [8]. Software system evolvability, incorporation of domain knowl-

edge in all phases of the software product-service life cycle is one of the trends [8].
Beyond this, countless opinions highlight that too much resources (e.g. intelligence, money, etc.) are spending today in developing software-service solutions that are more or less redundant (e.g. a significant amount of public funds – including national and European research and socio-economic supporting programs - has been spent on developing similar solutions). Moreover, criticism is directed on the fact that majority of these software-service solutions incorporate too many features and functional units than necessary at a certain moment for the user. Here, a more reliable approach is having the capability of configuring a customized software-service solution to the specific current need of the user by integrating in a fast and cost-effective way existent software-service units (modules) from the market (if they already exist), as well as by effective-reconfiguring the solution when the need is changed. This means, the user will pay only for what he/she really needs and uses now; no more, no less. To these, one should see that in an ill defined, turbulent and highly evolving business environment it will be almost impossible to have a complete definition of requirements in the development phase; new and significant requirements will be revealed later on, in the run-time phase of the system. But any requirement occurring in any stage of the run-time phase will have to be solved in real time and with affordable costs; in most of the cases by instantaneous integration of already available functionalities – doesn't matter which is the origin of the solution and what it is behind. Altogether, these contexts claim for a paradigm shift in conceptualizing, defining and developing a software-service solution over its life cycle. A reliable approach in this respect deals with the development of highly reconfigurable software-service systems.

3 About the reconfigurability paradigm

The development of reconfigurable software is desirable from a variety of reasons, among which adapting applications to changing environment, supporting on-line software upgrades, extending base software system functionality with additional services [10] - nowadays the Internet being one of the major driving forces towards building reconfigurable software [8] [11]. Any software that is developed as a reconfigurable system is inherently reusable.
When talking about software reconfigurability, the building blocks are the autonomous modules and interfaces. Conceptually, modules represent a separation of concerns, enforcing logical boundaries between components, such that none or few modules depend upon other modules of the system. In a reconfigurable design approach, to have as few dependencies as possible is a major importance issue. Modules are typically incorporated into a program through interfaces. A module interface expresses those elements that are provided and required by the module.

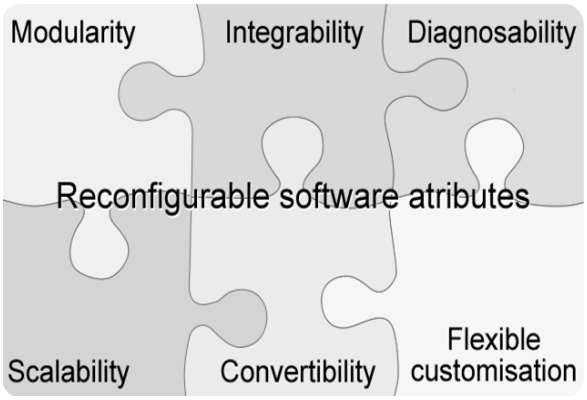

Fig. 1. Characteristics of reconfigurable software systems

Another key aspect is that when creating a design based on separate modules, instead of creating a monolithic application where there is no logical separation between components, all being gathered in a single but extended piece of code, one creates several smaller modules which, when composed together, will create the whole system. This makes modularized designed systems, if conceived right and developed accordingly, far more flexible than traditionally monolithic design since all or many of these modules may be reshaped and reused depending on the changing environment.

Reconfigurability goes beyond modularization. The main characteristics that set apart a reconfigurable software system from other software systems are: flexible customization, convertibility, scalability, modularity, integrability, diagnosability (see Figure 1).

Flexible customization: The modules are designed with application-oriented requirements to ensure precise level of flexibility for being able to support a "family of roadmaps" (family of task-packages). For a software system, flexible customization ensures wide cost savings over the life cycle. Flexible customization makes each dominant feature of a certain family of roadmaps determining system's configuration.

Convertibility: Changing from one roadmap to another one has to be done quickly. This feature is called convertibility and must be provided at module and system level. Quick conversion must be done either automatically or manually. Through the conversion process, modifications may occur in the structure of the modules as well as in the structure of the system, functionality may change, and a rapid calibration of the system may take place. Conversion is also used to add new modules to the system, that may affect or not its overall functionality and structure.

Scalability: Scalability is the counterpart of convertibility, which comes in and deals with system fine-tuning in small steps for increasing productivity (from the user's point of view). Fast and easy addition of system modules or additional functionality to existing modules characterizes this process. Expandability is also part of scalability: existing hardware can be upgraded or new hardware or software can be added to the system without reprogramming the software application.

Modularity: All major components of the software system are modular. Modular components can be easily replaced, maintained, improved, and updated. A module (and hence the technology implemented within that module) can easily be transferred to other users, or target groups, which are also using that framework, thus enabling an easy technology transfer. Modules should incorporate specific properties that make them ideal candidates for system reconfiguration, such as: code-related descriptions, description of the information passing through their interfaces, intrinsic flexibility, etc. Modules are reconfigurable only if their design and implementation is both independent of the target application and independent of the target hardware configuration.

Integrability: Clear rules of configuration and integration must be set up. Modules must be designed to be easily integrated into other systems.

Diagnosability: This characteristic includes two important aspects: detecting nonconformities and defects in the system and identification of causes for poor quality in operation. The design of diagnose-enabled software systems requires integration within modules and interfaces of systematic monitoring methods, as well as intelligent agents (for rapid and timely identification of errors).

It should be also noted that reconfigurability is in direct conflict with deploy-ability [12]. Both are needed, thus the challenge is to design a reconfigurable software system that is at the same time easily and timely deployable.

4 Methodology

For handling and implementing reconfigurability as a key performance characteristic of a software-service solution, a methodology for planning and deploying the reconfigurability function is further proposed. It consists of the following steps:

Step 1: Rank the constitutive dimensions of reconfiguration (modularity, convertibility, flexible customization, scalability, diagnosability, system integrability) using the AHP method and consistency analysis [13].

Step 2: Formulate key requirements in relation to each dimension of reconfiguration and rank them using AHP and consistency analysis.

Step 3: Determine key metrics for each dimension of reconfiguration.

Step 4: Determine the local value weight of each key metric by deploying local key requirements into local key metrics. Use a QFD-type philosophy in this respect [13].

Step 5: Determine the global value weight of each key metric by aggregating information from step 1 and step 4. Aggregation is done by multiplying the local value weight of each key metric with its parent's weight and the array obtained in this way is afterwards normalized.
Step 6: Use information from step 5 to select those metrics which play the most significant role in the equation of system reconfiguration.
Step 7: Identify the correlations between the metrics belonging to the subset selected at step 6. Extract all pairs of metrics that are negative correlated.
Step 8: Formulate inventive vectors of innovation for all pairs of conflicting problems identified at step 7. Use TRIZ method to solve this issue [13].
Step 9: Use information from step 8 to formulate generic guidelines to be followed during conceptualization, design and coding of software systems (including architecture, flows, processes, information, etc.) in order to build them for effective and efficient reconfiguration.

5 Guidelines on software development from the methodology application

The first step in the methodology described above requires ranking the constitutive dimensions of reconfigurability. In this respect, the common agreed perspective is that convertibility, integrability and scalability are the driving characteristics of reconfigurability, having the same impact in this equation. The AHP application leads to the results illustrated in Figure 2.

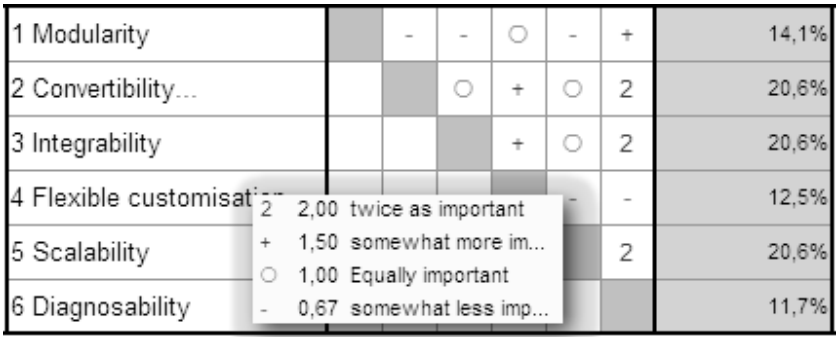

Fig. 2. AHP for ranking the objective-functions of reconfiguration

For consistency analysis of the results, an index *IR* depending on the number *n* of compared elements and the maximum eigenvalue λ_{max} of the system $\det[A_{n\times n} - \lambda \cdot I_n] = 0$ is calculated ($IR = |(\lambda_{max} - n)/(n - 1)|$) [13]. The pair-comparison is consistent if $IR \leq 0.1$. For the case in Figure 2, $\lambda_{max} = 6.05$, $n = 6$, thus $IR = 0.1$ (analysis is consistent).
The key requirements and their local importance for each objective-function, as they came up from the second step of the methodology are further introduced.
Modularity: high variety of tools developed in independent libraries (31.9%); as many as feasible interface between modules defined outside the modules (46%); multimodal interfaces (22.1%).
Convertibility: use of various roadmaps (application flows) (46.3%); rapid switch between different roadmaps (29.2%); rapid recalibration (bringing to nominal performance) (24.5%).
Integrability: documented code (32%); compatibility with different software technologies (18.5%); easy to integrate (low expertise required for integrator) (29.7%); rapid integration (19.8%).
Flexible customization: low cost over life cycle (28.6%); requirement-oriented flexibility (28.6%); cover dominant characteristics of roadmap's families (42.9%).
Scalability: rapid adjustments for new (change in) functionalities (46%); small scaling increment (22.1%); inexpensive adjustments (31.9%).
Diagnosability: presence of error handler (30%); error prevention (20%); comprehen-

sive monitoring (20%); testing of input data for validity (30%).

A set of 28 metrics have been identified during the step 3 of the methodology, distributed as follow: modularity: 4; convertibility: 3; integrability: 5; flexible customization: 4; scalability: 5; and diagnosability: 7. Figure 3 illustrates the weighting process of the metrics related to flexible customization. A similar process is performed to weight the key metrics for the other five objective-functions.

Optimization						
CTQs / Needs	1 Estimated average cost...	2 Average % of unused fu...	3 % of dominant characte...	4 Estimated cost of maint...	Number of significant relations...	Importance to Customer
1 Low cost / life cycle	⊙	○	○	⊙	4	28,6
2 Requirement-oriented fle...		⊙	○		2	28,6
3 Cover dominant charact...			⊙		1	42,9
Number of significant relationshi...	1	2	3	1		
Importance	18,2%	24,2%	39,4%	18,2%		

Fig. 3. Weighting the key metrics that characterize flexible customization

The global weights of the 28 metrics come up by weighting local results with the ranks of objective-functions. Because of the limited space in the paper, some of the most impacting metrics and their global weights are further introduced for exemplification: (a) in relation with convertibility: no. of different roadmaps (10.6%); time/switch (8.4%); (b) in relation with scalability: time for adjustments (inexpensive adjustments) (7.2%); adjustment time/new functionality (5.6%); (c) in relation with integrability: expertise level: novice(1-2) senior(3-4) expert(5) (5.2%); % of documented classes (4.8%); time for integration (4.2%); % of documented functions (4.1%); (d) in relation with flexible customization: % of dominant characteristics that are covered (4.9%); average % of unused functionalities / configuration (3.0%); (e) in relation with modularity: no of different technologies for interface (4.3%); % of interface defined outside the module (3.9%); granularity level (no. functions/tool) (3.0%); no. different tools/library (2.9%).

This set of 14 metrics above presented represents the set of most impacting metrics in relation with reconfigurability (the major metrics). The other 14 metrics (not presented here because of the limited space of the paper) have weights ranging from 1.3% to 2.7%, they bringing only about 28.2% impact in the equation of reconfigurability. One could identify this set of less impacting metrics includes all metrics belonging to diagnosability, three metrics of scalability, two metrics of flexible customization, one metric of integrability and one metric of scalability from the initial set of 28 metrics.

The correlations between the most relevant 17 metrics, as they are established based on their global weights, are summarized in Figure 4. In this set of correlations, 11 pairs of negative correlated metrics are revealed. They represent key challenges in developing mature reconfigurable software solutions.

To approach innovatively these conflicts, TRIZ method has been applied [13]. According to the TRIZ algorithm, the 11 pairs of conflicting metrics are translated into pairs of conflicting TRIZ parameters. Using the TRIZ contradiction matrix, a list of generic inventive vectors is determined. They represent

"reference systems" in formulating innovative solutions for the problem under consideration.

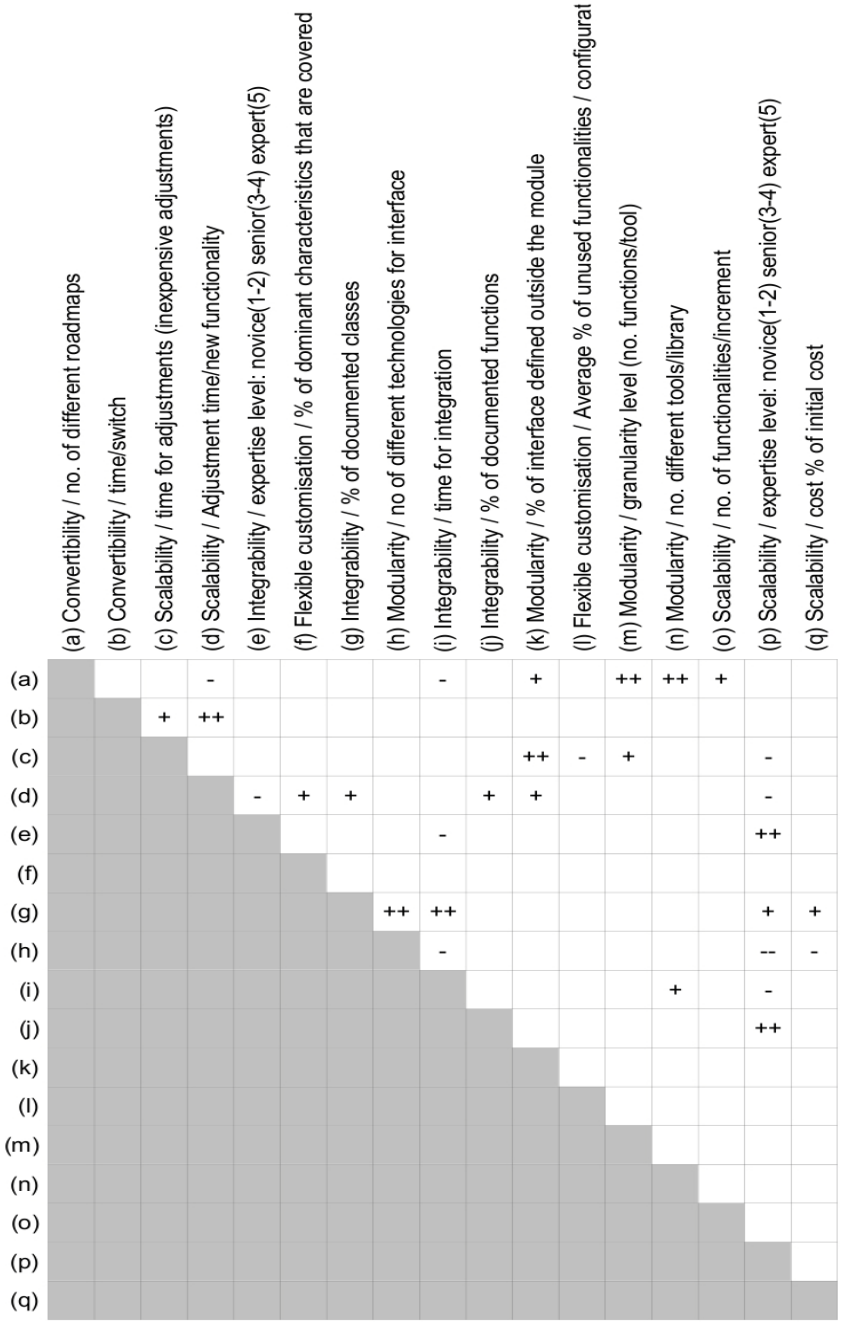

Fig. 4. Correlations between the most relevant metrics

Thus, the following general recommendations to support effective and efficient reconfiguration have been generated:

- Use a work breakdown structure for a large project – the reconfigurable software methodology might be based on RAD or prototyping model,
- In the design stage, separate the part of the application dealing with multiple interfaces from the logic part of the system, e.g. use a three layer (tyre) architecture,
- In the analysis stage, each tool to be provided or every function that builds up a specific logical module should be planned as versatile as possible, but the designer might want to consider also the overall versatility of its parent tool or of the entire

system when designing a specific function or tool,
- When designing the software, similar functions should be merged together in one or more versatile functions. Similarity might not be obvious at all, that's why emphasis should be directed towards analysis stages. People with abstraction and strong communication abilities should be included in the analysis team,
- Build requirements also against the possible future customer demands. Although this is a difficult task, a good practice is to know your user as well as possible before the analysis stages start,
- Build specific functions in such a way that their input data is computed before you call these functions. Keep a good balance between such computations and redundant data,
- Build interfaces or communication rules between physical modules or libraries to be as technology-independent as possible. Stick to open source technologies if possible, as they provide higher portability,
- Use a plug-in approach, but don't make plug-ins technology-dependent. For example, don't build them to rely on a specific operating system. Move technology-dependent functions into a separate module for which you might provide separate versions for each platform,
- Provide skins and/or templates also for the user interface,
- If it's feasible, create an open architecture system, give access to source code,
- Build advanced settings panels, as these might solve specific reconfigurability issues. Use presets when possible. Also allow each user to store the settings in his/her profile, so that each user might customize the application as he or she wants,
- Use a presentation layer in order to reduce interface code intervention when adding new business functionality with the users or vice versa. Also use the presentation layer to separate the logic of the system from the devices (technologies) used to interact with it.

These guidelines should be added to those derived from the base development methodology chosen for the software.

6 Case study: web-based software platform for product innovation

A. Project context

Since innovation was placed in the centre of economic development theory, a lot of product innovation models have been developed (according to some opinions, up to 17). Some of them are incremental models; some others are top-down or bottom-up models [14]. However, beyond the theoretical models of product innovation practice reveals a huge variety of innovation roadmaps, depending on local conditions [15]. Moreover, innovation is a dynamic process [15]. Under such circumstances, a web-based software platform for supporting product innovation is an excellent candidate for a development approach which considers reconfiguration a key performance indicator. The authors' experience in this direction is further revealed in the next sections of the paper.

B. Project description

In this section, we will briefly introduce a case study for a reconfigurable software application (TECH IT EASY) that is implementing a high diversity of conceptual models related to product innovation management.

The main goal of TECH IT EASY is to smartly assist users in (re)defining (innovative) products and/or processes within their companies, incorporating in the same time intrinsic resources for fast and cost-effective reconfiguration to specific application-related needs (including integration of specialized external modules: e.g. ontologies for web and database search).

Even if TECH IT EASY is a supporting tool for the innovation processes within a company, it does not automate in any way these innovation processes. While it's final version will provide several roadmaps for innovation, these will not be able to replace human capacity for creativity, but rather support it.

The TECH IT EASY system contains three groups of tools: modelling tools, search & analysis tools, and knowledge base tools. According to the reconfigurable software requirements, each tool is developed as a separate module which communicates with the other modules by means of interfaces.

The software application is designed having in mind two distinct types of actors: moderators and analysts. The moderator will be able to create a project and grant different permissions to analysts for a certain project. In order to cope with different business cultures, thus different environments (which are the playground of reconfigurable software systems), the moderator may act, according to the company policy, as a project supervisor, administrator, moderator, project owner or facilitator.

An analyst is an expert involved in the innovation process. In order to work on a project or to see the project, the analyst needs to have the permission from the moderator.

In order to adapt to different innovation patterns or routines, but also to cope with possible novice innovators, the project is created by the moderator as a roadmap-driven innovation project. The roadmap is defined using an innovation ontology.

A roadmap-driven methodology, in the TECH-IT-EASY tool framework, consists of specific activities that are based on several systematic methods and techniques of innovation management (e.g. this time, QFD, TRIZ, Su-Field analysis, and Laws of Evolution modules are included in the system), which might or might not be used within an innovation project.

For each activity, the ontology defines the specific tools that are to be used in order to complete the activity/task. For a certain activity the analyst may choose from an array of tools, all of them described in the ontology as being tools that support the completion of that activity.

The ontology describes conditionality relations between activities. In this way the system gives an overview to the user of what activities were performed already, and what activities may be carried out in the future based on the past activities already completed.

One key issue, which calls for a reconfigurable approach in the TECH IT EASY tool, is that in the near future new innovation tools might be added (as distinct modules) and new roadmaps might be used, by adding new innovation roadmaps with the help of new innovation ontologies.

Although this aspect was not initially requested within project's specifications, it came up later on following the guideline stating that "requirements should be built against the possible future customer demands".

The roadmap-driven innovation project can be tailored to point out (graphically and textually) the steps required by a certain innovation methodology. After completing a certain step, the next step in the methodology sequence will be pointed out as the next task to be completed. The analyst may choose at any time to leave the roadmap. This concept offers a balance between the flexible (open) and fixed innovation methodology approaches.

C. Reconfigurability issues for the TECH IT EASY tool

Because in this project requirements were not clear at the beginning of the analysis stage and will probably remain "open to changes" even at the end of the development stage, reconfiguration-oriented design of the TECH IT EASY tool is highly desired. Therefore, the planning process was based on the general guidelines issued in section 5. This means, a living prototype approach was considered for managing the life cycle of this software tool.

Regarding technologies, Java, JSP, HTML and MySQL were chosen, because they are wide-spread, well-known and are suitable to support reconfigurability paradigm. Having in mind one of the general recommendation, we used only open source technologies, e.g. The Spring Framework was chosen because it is a powerful open source application framework for the Java platform [16]. Spring offers a good ratio in terms of productivity over the learning curve. These technologies are also suitable to solve portability issues,

thus their use enhances the reconfigurabili- tyof the TECH IT EASY application.

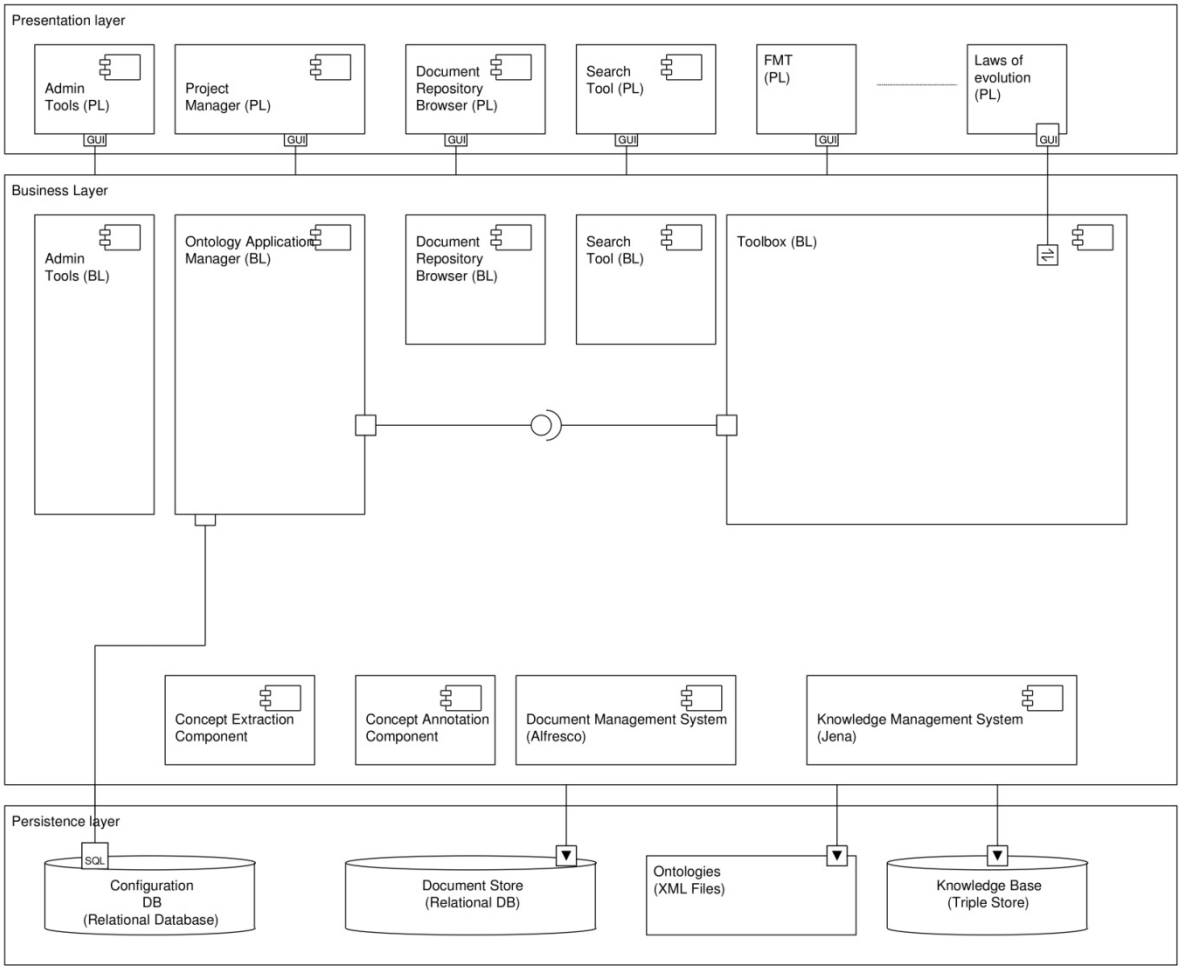

Fig. 5. The three tire application architecture of TECH IT EASY

As the general recommendation obtained from TRIZ inventive vectors has indicated the use of three-tier architecture, TECH IT EASY was designed starting with this in mind (see Figure 5). Also, following the guidelines, we separated the presentation layer from the business logic, one of the advantages being that the user interface may be customized by applying templates.

Actually, the popular-technology choice, Spring and MySQL, was made by evaluating the innovative vector no. 3 from the TRIZ conflict matrix [13], while trying to solve the negative correlated metric-pair [“no of different technologies for modules” of modularity vs. “expertise level” of scalability].

The core of the application is the Ontology Application Manager. We conceived this module having in mind the user of the TECH IT EASY application system is not a trained innovation annalist, and thus he needs guidance in finding the right steps in the innovation process. This module gives an overview on the current status of the project, as well as on which activities are next on the line, as well as on which tools are to be used to complete those activities. As said before the behaviour of module relies on the ontology that describes the innovation process, ontology that is stored in the Ontologies module from the Persistence Layer. The innovation methodology may be changed by changing the innovation ontology. The Ontology Application Manager offers also a percentage approximation of the progress of the current innovation project.

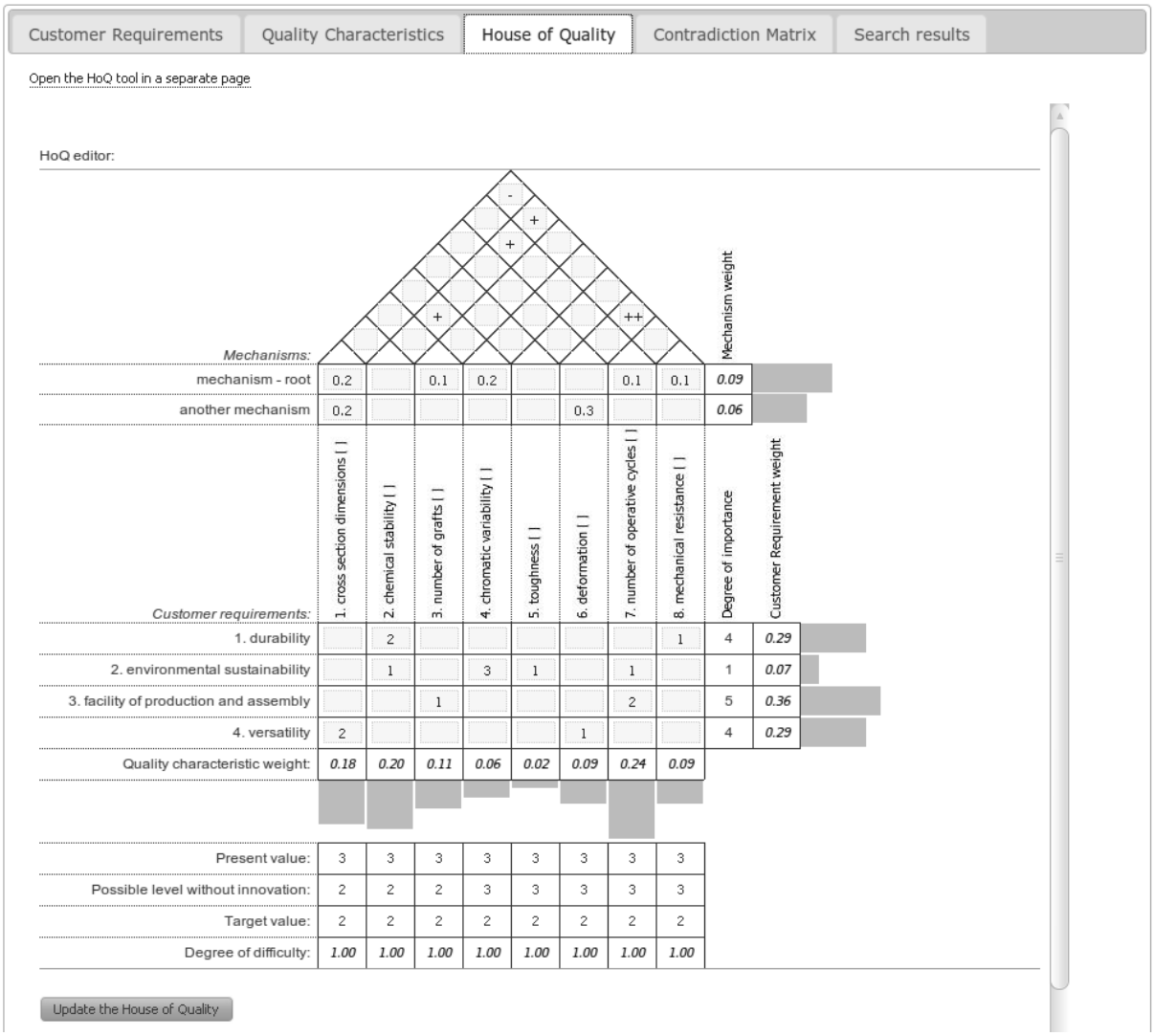

Fig. 6. The House of Quality tool as seen in the TECH IT EASY application

From a conceptual point of view, for each activity, a tool (see Figure 6) or a set of tools may be called from the Toolbox module. From a programmer point of view the toolbox is an interface that each tool has to implement. We choose to use a toolbox in order to offer a common interface for the Ontology Application Manager when calling various tools. In this way new tools may be added at any time, the only condition is that they must implement the Toolbox interface. In order to add a new tool a developer has to create also the corresponding module in the presentation layer, in other words the user interface for that particular tool.

In order to further increase the degree of reconfigurability, we turned the attention to the information (knowledge) storage. To facilitate an open communication between any technology that may be part of the Persistence Layer architecture, and the various modules present in the Business Layer, we choose to develop two dedicated modules, the Document Management System and the Knowledge Management System.

The Document Management System is offering a consistent interface for each module (e.g. Document Repository Browser) so that they can access different Document Repositories technologies (e.g. Alfresco DMS).

The Knowledge Management System is responsible for creating a bridge between the ontologies (e.g. Triple Store) and the modules that need to access the information stored in them (e.g. Ontology Application Manager).

As the technologies chosen are quite popular, although the application will support a “mix”

of technologies, future module developments are more likely to employ the already-in-use ones. Besides that, it's also much more easily to get expertise on a popular wide spread technology.

As a note, choosing well-known technologies for developing a software application doesn't always mean they're the best choice for attaining technical excellence or even for obtaining average performances. Actually, when elaborating the vision document for the TECH IT EASY tool, the technologies that analysts first thought of were a mix of a web server with a desktop client (using FreePascal or Delphi). But as the in-deep analysis revealed, the aim is obtaining only a reasonable level of performance while solving the negative correlated metric-pair above.

In this framework, requirements are permanently assessed and enhanced by a planning team consisting of the technical management, developers, researchers on innovation methodologies, researchers on ontology-based searches, and users.

As innovation methodologies are to be applied by a team, emphasis was also placed on communication issues among users that collaborate in a specific innovation project. Although this was not a requirement, future communication aspects that would possibly emerge after users will start working with the application were assessed.

The source code for all classes can be made available, allowing customers or potential users to fine-tune the application if they consider so.

7 Conclusions and further researches

Using the proposed methodology we successfully managed the innovative design process of the TECH IT EASY tool. Its innovativeness resides in supporting a variety of flexible application-related methodologies, even those for which new tools will be developed in the near future.

A good understanding of requirements, superior communication in the planning team, combined with the "know-your-user" approach in mind and with visual class libraries, APIs, dynamic link libraries and a good IDE might be a feasible counterpart to reconfigurability issue in software development.

Acknowledgements

Financial support from the European Commission within the FP7 research project TECH IT EASY/232410 is acknowledged with gratitude.

References

[1] O.J. Dahl, E.W. Dijkstra and C.A.R. Hoare, "Structured Programming," Academic Press, London, 1972.

[2] R.T. Futrell and D.F. Shafer, L.I. Safer, "Quality Software Project Management," Prentice Hall PTR, 2002.

[3] T. Gill, "Planning Smarter: Creating Blueprint-Quality Software Specifications," Prentice Hall PTR, 2002.

[4] H.S. Ismail, S.P. Snowden, J. Poolton, I.R. Reid and I.C. Arokiam, "Agile manufacturing framework and practice,"*International Journal of Agile Systems and Management*, 2006.

[5] P. Kettunen, "Adopting key lessons from agile manufacturing to agile software product development – A comparative study,"*Technovation Journal*, 2009.

[6] T. Dyba and T. Dingsoyr, "Empirical studies of agile software development: A systematic review,"*Information and Software Technology Journal*, 2008.

[7] M. Abrahamsson, J. Warsta, M.T. Siponen and J. Ronkainen, "New directions on agile methods: a comparative analysis," in *Proceedings of the 25th International Conference on Software Engineering*, 2003.

[8] ***, CORDIS ICT Programme, "Objective 1.2: Internet of Services, Software and Virtualization," http://cordis.europa.eu/fp7/ict/ssai/objectives-1-2_en.html, accessed at 16.09.2009.

[9] M. Papazoglou and K. Pohl (ed.), "Longer term research challenges in

Software & Services,"*Expert Group Report for EC*, 2008.
[10] K. Whisnant , Z. T. Kalbarczyk and R. K. Iyer, "A system model for dynamically reconfigurable software,"*IBM Systems Journal*, v.42 n.1, p.45-59, January 2003.
[11] Z. Chang, X. Mao and Z. Qi, "Towards a Formal Model for Reconfigurable Software Architectures by Bigraphs; Software Architecture," in *the Seventh Working IEEE/IFIP Conference*, Volume 17, Issue 18-21: 331–334, DOI 10.1109/WICSA. 2008.
[12] S. Kamin and L. Clausen, "Dynamically reconfigurable software components," University of Illinois at Urbana-Champaign 1304 W. Springfield Urbana, October 31, 2001.
[13] S. Brad, "Complex System Design Technique," Dacia Publ., 2008.
[14] J. Tidd, "From Knowledge Management to Strategic Competence: Measuring technological, market and organizational innovation," Imperial College Press, 2006.
[15] J., Tidd, J., Bessant and K. Pavitt, "Managing Innovation: Integrating technological, market and organizational change," Wiley, 2005.
[16] D. Minter, "Beginning Spring 2, From Novice to Professional," Apress, 2008.

Stelian BRAD is full professor at the Technical University of Cluj-Napoca, Romania, leading the research group on Competitive Engineering in Design and Development. He is also the Director of the Department of Research, Development and Innovation Management of the same university. His research interests include competitive engineering, engineering and management of innovation, intelligent industrial robotics.

Adrian CHIOREANU is a senior researcher at the Technical University of Cluj-Napoca, as a member of the research group on Competitive Engineering in Design and Development. His key professional fields of interest are: information technology and systems in business, multimedia, image processing. Adrian got his PhD in telecommunication in 2009. Currently he's part of a Post-PhD program and he's learning to get his ACCA qualification.

Mircea FULEA is PhD student and researcher at the Technical University of Cluj-Napoca, Romania, member of the research group on Competitive Engineering in Design and Development. His professional key interests are product management, software design and development, graphic design and multimedia applications.

Bogdan MOCAN is assistant professor at the Technical University of Cluj-Napoca, Romania, member of the research group on Competitive Engineering in Design and Development. His key professional fields of interests are process and product innovation, robotics, integrated management systems and quality management.

Emilia BRAD is lecturer at the Technical University of Cluj-Napoca, Romania, member of the research group on Competitive Engineering in Design and Development. Her research fields include production planning and flexible manufacturing systems.