2010

A Model-Driven Development (MDD) Approach to Change Impact Analysis

Fabio Fabio Marzullo
Federal University of Rio de Janeiro Centro de Tecnologia, fpm@fabiomarzullo.com.br

Vitor Failace De Mario
Federal University of Rio de Janeiro Centro de Tecnologia, vdemario@ufrj.br

João Paulo da Silva
Federal University of Rio de Janeiro Centro de Tecnologia, jpaulo@ufrj.br

Leonardo Santoro Nunes
Federal University of Rio de Janeiro Centro de Tecnologia, sleo@cos.ufrj.br

Jano Moreira de Souza
Federal University of Rio de Janeiro Centro de Tecnologia, jano@cos.ufrj.br

A Model-Driven Development (MDD) Approach to Change Impact Analysis

Research-in-Progress

Fabio Perez Marzullo
Federal University of Rio de Janeiro
Centro de Tecnologia, Block H, suite 306
fpm@fabiomarzullo.com.br

Vitor Failace De Mario
Federal University of Rio de Janeiro
Centro de Tecnologia, Block H, suite 306
vdemario@ufrj.br

João Paulo da Silva
Federal University of Rio de Janeiro
Centro de Tecnologia, Block H, suite 306
jpaulo@ufrj.br

Leonardo Santoro Nunes
Federal University of Rio de Janeiro
Centro de Tecnologia, Block H, suite 306
sleo@cos.ufrj.br

Jano Moreira de Souza
Federal University of Rio de Janeiro
Centro de Tecnologia, Block H, suite 306
jano@cos.ufrj.br

Abstract

This paper presents a change impact analysis approach created as part of a model-driven development (MDD) environment. Its purpose is to give project stakeholders the ability to better assess the impact that requirement changes will impose on project schedule and budget. Using MDD techniques and visualization tools we were able to automate requirement change impact analysis. Therefore, the following discussion focuses on how this environment can help software project managers, developers, and other stakeholders increase their productivity and evaluate how client changes affect their daily work.

Keywords: MDA, change impact, domain models, UML, project management, change management.

Introduction

Model-driven development (MDD) and model-driven architecture (MDA) approaches are now a reality (Frankel 2003; Rodrigues 2004) and, as they reach the status of those important tools that are used in any development project (Marzullo et al. 2008), challenges faced by IT managers have only increased. Due to painstaking, detailed domain-specific functionalities and ever-growing knowledge consumption, investment has become necessary in order to improve the usability of such techniques and minimize the time and resources spent in software projects.

As a consequence, business processes require more agile and flexible Information Technology (IT) assets that can be easily and swiftly customized to meet their changing demands. Such challenges are continuously faced by IT professionals as they are required to develop innovative solutions to cope with business-related problems. In a dynamic business environment, innovation is a way to create value and it may result from the combination of requirement engineering and model-driven development (MDD) approaches.

This is the scenario we have been facing along the past three years, where we have been working on a MDA/MDD research project environment named MDA4Eclipse (MDA4Eclipse 2010), and have faced many such challenges to create an environment that would improve our daily work as developers.

MDA4Eclipse is currently in its early stages and aims to create a set of development tools capable of controlling software development life-cycle, modelling, development, test and maintenance, and consists of the following modules:

1. The MDA/MDD module, responsible for reading the business models and generating the application code;
2. The assessment module, responsible for controlling the development cycle, as regards requirements, change impact, software code measurement, and test cases artefacts.
3. Domain-sharing module, responsible for controlling the distributed and cooperative domain sharing development.
4. Service-driven development module, responsible for maintaining the MDD as a service approach.

In this study, however, we are focusing on module two, which is responsible for (amongst other things) analyzing the impact related to project requirements changes. Our goal was to prove that change impact analysis (Bohner et al. 1996) can be automated by using MDD/MDA techniques and by embedding proper information in conceptual models. We also wanted to create a traceability infrastructure capable of highlighting the impact a certain requirement change would impose on the code and how changes in the source code would affect its associated requirement.

Several technologies came together to help us pursue this goal. In this study we use Unified Modelling Language (UML) – as implemented by ArgoUML version 0.28.1 – to create all conceptual models, and use tagged values as a mechanism to embed metadata in the system design (Pressman 2005).

The general idea was that usual elements such as classes, packages, methods, properties, relationships, when properly tagged, should be able to 'tell' the MDD engine (MDA4Eclipse 2010) which requirement(s) that element represents, and consequently create annotated source code (Sun 2010) that would be then eligible for analysis.

Annotation works as a tagging mechanism and can be used to trace any source code artefact that is created to fulfil a certain requirement. It is the core intelligence of the analysis approach as it helps create graphs showing the impact requirement changes inflict onto classes and other requirements.

The following sections highlight the motivations and the conceptualization to create such change impact analysis approach, and how it can be applied to most development projects. Finally, we demonstrate how the proposed tool should be used to help with impact analysis, and conclude with a look at paths for future work.

Motivation

The Brazilian Navy and the Database Laboratory at the Federal University of Rio de Janeiro created in 2007 a taskforce to use a new development approach to improve a set of management practices applied to software projects. After shifting from traditional development paradigms to MDD, we obtained important results regarding productivity and project cost (Marzullo et al. 2009, Masticola 2007). To illustrate this, we have cut our development time by approximately 20%, gaining more efficiency in our development process without affecting requirement quality.

In any development project, managers have difficulties in predicting the actual impact the changes in any requirement will have on the development process, or how changes done to the code would affect its equivalent requirement.

So, to address this problem we began by stating three topics that needed to be addressed:

1. How to come up with a MDD approach to help mapping requirements with source code artefacts?
2. How to automate the process of creating such map?
3. How to analyze the impact related to requirements changes?

The following sections attempt to answer each question by detailing what we have done to create such infrastructure.

Change Impact Approach

Regardless of the development process involved, the pursuit of a systematic impact analysis approach which efficiently enables managers or other stakeholders to extract critical software-related information still exists in software projects (Rajlich et al. 2004). Few techniques are available to help developers in doing this (Law et al. 2003); and, to do something to contribute, we have created an impact analysis module capable of reading source code and creating impact graphs that show the relationship between requirement and code.

Some professionals may say that this is not an innovative idea. Possibly true. However this is a simpler, cheaper and more effective approach then some presented by other commercial tools like MindTools (MindTools 2010), Celadon (Zhang 2008), and Rational Software Architect (IBM 2010). The approach presented here is different as it demonstrates that change impact analysis can be straightforward when using the Model-driven development paradigm. Unlikely the IBM toolset that requires different tools to accomplish the link and synchronization from requirements to models, ours allows direct association between requirement – model – code and creates a traceability path that can be visualized by a set of hyperbolic tree graphs. Also when using the IBM software one needs to manually maintain this synchronization whereas in our approach the synchronization is done whenever the software is generated and the only human intervention happens at design time. So, the bottom line is that the innovation is associated with the use of MDD techniques that simplify the link and synchronization, making the analysis easier by building an internal code traceability structure.

Therefore, the purpose of this approach is to probe the entire source code tree and search for requirement annotations that were embedded by the MDD engine (the annotation is placed in code by the MDD engine as the analysis model is tagged, at design time, with the identification of the requirements) and to create a visual graph capable of displaying the actual impact path. Such graphs can be presented in two perspectives: (1) by requirement dependency, which presents a high level source of information; and (2) by class dependency which presents a low level source of information. Both impact graphs can be presented to clients in order to help them decide whether the changes should be carried out or not.

The analysis strategy works in three ways:

(1) from requirement to code, which shows how a requirement change will affect the source code;

(2) from code to requirement, which shows the requirements that a source code update will affect; and

(3) from requirement to requirement, which shows, for example, if a change in requirement R(i) will affect requirement R(i+n).

A conceptual example can clarify the first strategy: suppose we want to check whether changes in requirement R1 will impose any effect on the source code. To create the proper infrastructure, the analyst, at design time, must tag all classes that will fulfil requirement R1 with the `<<Requirement=R1>>` tagged value.

At development time, the MDA4Eclipse reads these tagged values and writes annotations in the generated class and in every other class or file that is needed to comply with the software architecture. By doing this, MDA4Eclipse guarantees that each requirement will be perceived as a set of source code elements (entity classes, view files, utility classes and others) and, whenever we want to see the extension of the impact that any change will inflict on the software, the analysis module reads the entire source code path and searches for the `@Requirement` annotation containing the correct requirement ID (in this case R1).

By looking at Figure 1, the class GenericPanel is part of requirement R1 and will be evaluated whenever changes are applied to it. As seen in the class diagram, the GenericPanel class is associated with both R1 and R2 requirements so the tagged value that will link this class with both requirements is viewed as `<<Requirement=R1,R2>>`. Finally, one can understand the impact by analyzing the hyperbolic tree graph shown in Figure 2. After completing the analysis it is possible to accurately decide whether to proceed with the change or abandon it.

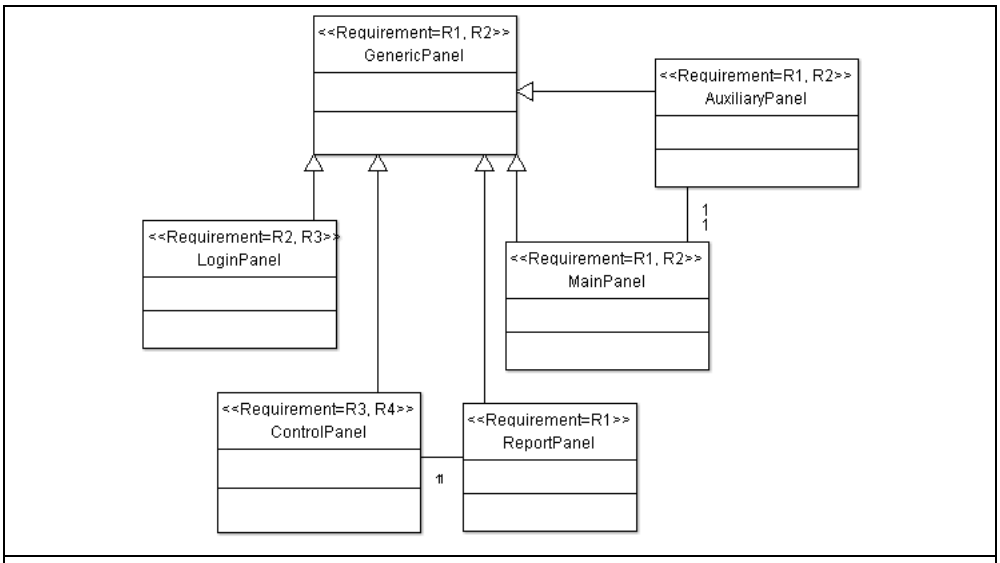

Figure 1. Example of a class diagram that is tagged to be associated with functional requirements.

One might say that requirement association should be done in lower level elements such as class methods or properties, however this first approach is meant only to understand which high level artefacts would be affected whenever changes occur. It is clear that the accuracy can be improved, and this will be tackled in the future.

Finally, aiming at creating a visual tool to facilitate the way stakeholders will evaluate change impact we decided to create a visual graph using a hyperbolic tree (Lamping et al. 1995). Next, whenever a requirement was subjected to change, the project manager could see the classes affected by this change in a much more meaningful way.

The capabilities of the hyperbolic tree give not only the project manager but also the developer flexibility and understanding of what will have to be done when proceeding with the change.

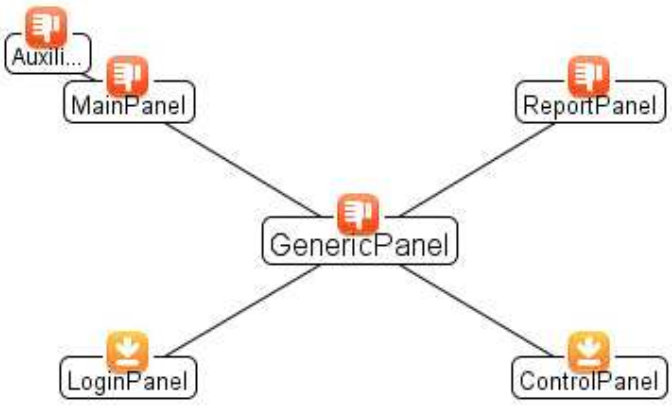

Figure 2. Conceptual example of an impact analysis in a low level view (class level view) from Figure 1 (image created with the Treebolic browser) (Bou 2009).

Each node is assigned an impact level mark. These marks represent a dependency level varying from direct impact to indirect impact. The hand-down icon means that marked classes are directly affected by requirement change, and the arrow-down icon means that impact on near classes might affect their associations.

At this point, impact evaluation is determined by knowing whether the class is contained in the requirement class set or not. A second approach (that will probably be the subject of a second paper) is under evaluation and will use coupling metrics to determine impact and dependency.

Change Impact Analysis in Action

Extending the conceptual example previously described, and in order to prove that our change impact analysis approach would perform as detailed, we decided to use it in a real-life project. This concept proof was based on a project conducted at the Brazilian Navy Bureau for Integrated Logistic Support (NALIM).

Currently, NALIM is developing a system to support maintenance logistics for the Brazilian Navy fleet. Our goal was to see whether this approach would help evaluating the impact that would ensue whenever a requirement update was detected.

Step-by-step, the analysis of changes applied to the *"Create a Service Request"* requirement (from now on named requirement R1) can be perceived as this: Initially, the business analyst, while creating a class diagram, embeds the tagged value `{mb.sisali.requirement}` with the corresponding set of requirements in each class. As shown in Figure 3, in order for the requirement to be associated with the generated code set, the analyst tags the class ServiceRequest with the `<<mb.sisali.requirement=R1,R2>>` (in this case the ServiceRequest class is also related to another requirement R2.

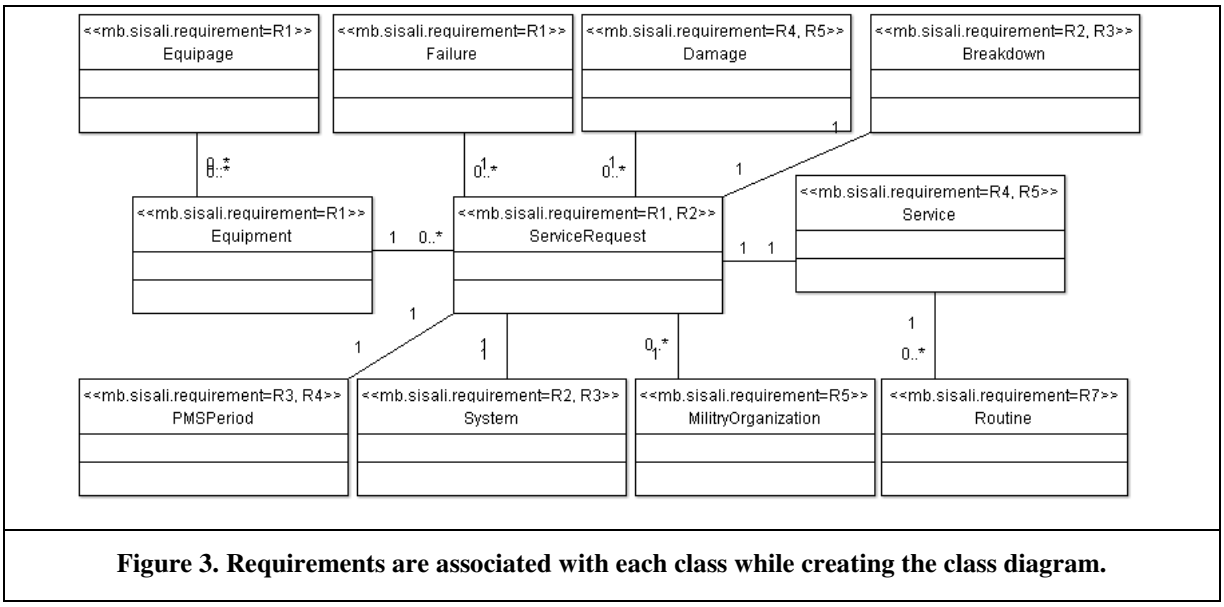

Figure 3. Requirements are associated with each class while creating the class diagram.

This class diagram was modelled using ArgoUML version 0.28.1 (ArgoUML 2010), and was exported as an XMI file (MOF 2009). Listing 1 is only an excerpt of the entire XMI file, as it would not otherwise fit this paper. The tagged value is highlighted and shows that class ServiceRequest is assigned requirements R1 and R2. From that information we can create the associated source code set with MDA4Eclipse.

```
<UML:Class xmi.id='1' name='ServiceRequest' visibility='public'>
  <UML:ModelElement.taggedValue>
    <UML:TaggedValue xmi.id = '2' isSpecification = 'false'>
      <UML:TaggedValue.dataValue>R1,R2</UML:TaggedValue.dataValue>
      <UML:TaggedValue.type><UML:TagDefinition xmi.idref='3'/>
      </UML:TaggedValue.type>
```

```
    </UML:TaggedValue>
</UML:ModelElement.taggedValue>
<UML:TagDefinition xmi.id='3' name = 'mb.sisali.requirement'/>
<UML:Classifier.feature>
  <UML:Attribute xmi.id='7' name='prioridade' visibility='public'/>
  <UML:Attribute xmi.id='8' name='estadoAtual' visibility='public'/>
  <UML:Attribute xmi.id='9' name= reparoABordo visibility='public'/>
  <UML:BehavioralFeature.parameter>
  <UML:Parameter xmi.id='14' kind='return'/></UML:BehavioralFeature.parameter>
  </UML:Operation>
  <UML:Operation xmi.id='15' name='getState' visibility='public'>
    <UML:BehavioralFeature.parameter>
      <UML:Parameter xmi.id='19' kind='return'/></UML:BehavioralFeature.parameter>
  </UML:Operation>
  </UML:Classifier.feature> {…}
</UML:Class>
```

Listing. 1. Extracted from the system domain model (XMI file). The XML excerpt shows how tagged values are linked to domain classes. (Identifications were altered to fit this article). File created by ArgoUML version 0.28.1.

Following, MDA4Eclipse reads the XMI file (listed above) and generates the system source code. When it reads the tagged value `{mb.sisali.requirement}` (which is referenced by the tag `<UML:TagDefinition xmi.idref = '3'/>)` it adds an annotation like `@Requirement("X")`, X being the requirement ID, (in this case acquired from the tag `<UML:TaggedValue.dataValue>R1,R2</UML:TaggedValue.dataValue>)` to every artefact that is generated and is associated with that class. Class 1 is an example of how the annotation is added to a class.

```
package com.core.beans.sisali;
import java.io.Serializable;
@Requirement("R1,R2")
public class PedidoServico implements Serializable {
    private String prioridade;
    private String estadoAtual;
    private String reparoABordo;
    public String getPrioridade() {return prioridade; }
    public void setPrioridade(String prioridade) {
       this.prioridade = prioridade;
    }
    public String getEstadoAtual() {return estadoAtual; }
    public void setEstadoAtual(String estadoAtual) {
       this.estadoAtual = estadoAtual;
    }
    public String getReparoABordo() {return reparoABordo; }
    public void setReparoABordo(String reparoABordo) {
       this.reparoABordo = reparoABordo;
    }
}
```

Class 1. ServiceRequest class. The `@Requirement` annotation is placed right above the class name.

Finally, whenever a stakeholder needs to acknowledge the level of impact a requirement update will cause the source code it is possible to generate the impact path, as shown in Figure 4. It details the impact from a low-level view, which means that the analysis is done at class level. Therefore, the impact is assessed on a requirement – class approach.

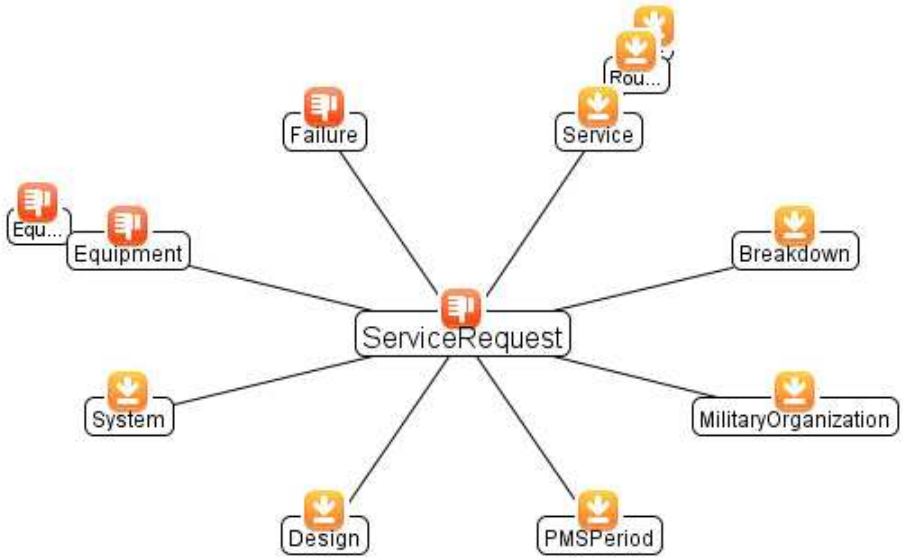

Figure 4. Low level change impact analysis view. Classes associated with "PedidoServico" are directly and indirectly affected by requirement change.

A second approach can be seen in Figures 5, 6, and 7. It is possible to analyze the impact from a high-level view, in a requirement-to-requirement perspective. In Figure 5 it is possible to understand that both the R1 and R2 requirements are associated to the ServiceRequest class, so if a change is made to requirement R1 it will affect the ServiceRequest class, therefore, as requirement R2 is also linked to the ServiceRequest class it is, by transitivity, propagated to requirement R2.

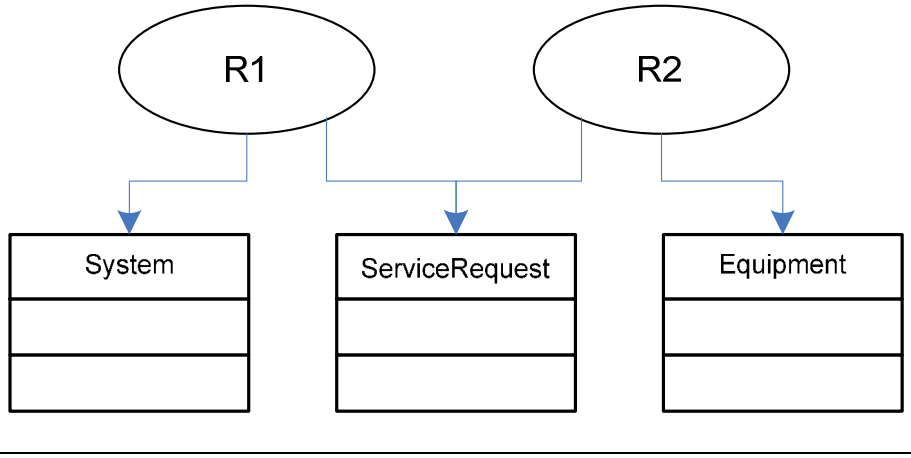

Figure 5. Requirement-requirement impact definition.

Figure 6 then shows the system work breakdown structure (PMI 2004) as a way of clarifying the system requirement structure. Figure 7 takes advantage of this organization to create the high-level view and create the hyperbolic tree graph. By using the approach explained previously, it is straightforward to assess the impact on a requirement-to-requirement level.

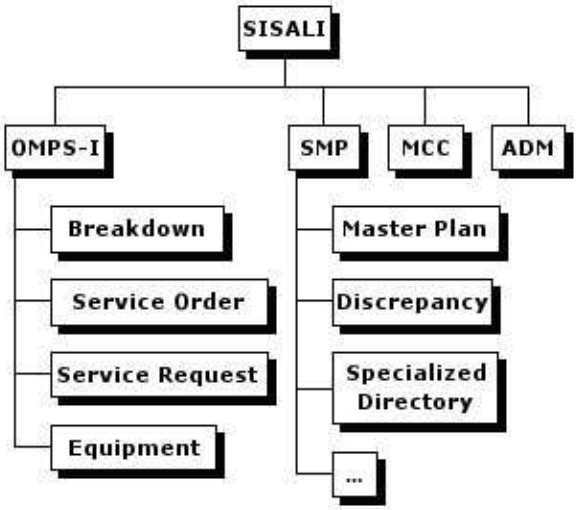

Figure 6. Sisali's work breakdown structure (WBS) (PMI 2004). The system (SISALI) can be viewed as four modules (OMPS-I, SMP, MCC and ADM), each consisting of a set of functional requirements such as "delineamento", "ordem de serviço", "pedido de serviço" (outlining, service order, service request), etc.

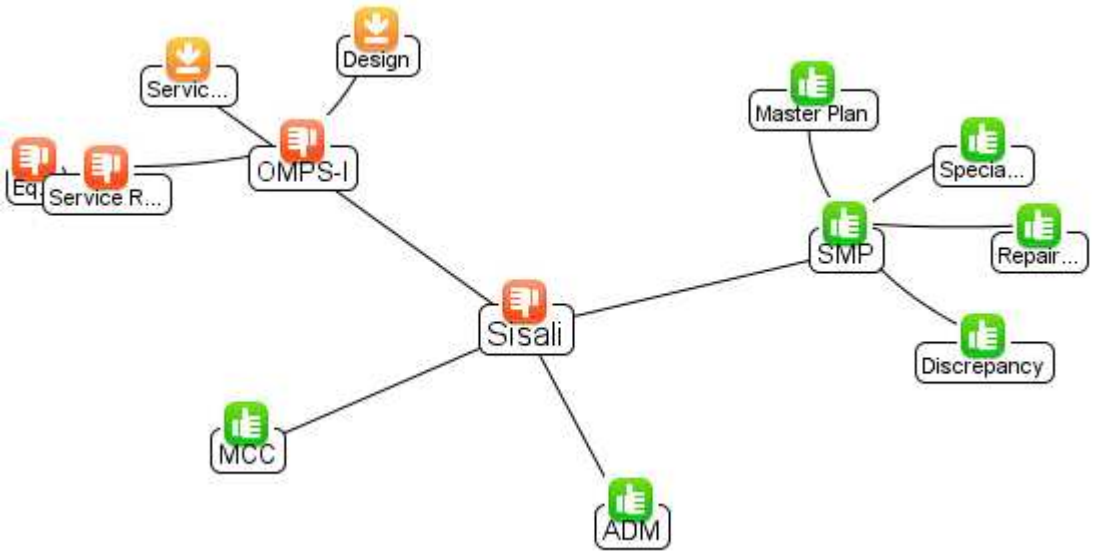

Figure 7. A second High-Level change impact analysis view is also possible. Based on the project WBS (Figure 6), changes to the "ServiceRequest" requirement will directly and indirectly affect other "OMPS-I" module-associated requirements. Green marks mean requirements are not affected by changes.

Preliminary Analysis and Results

Understanding and using the impact analysis tool is straightforward, as previously demonstrated, and at this point we present preliminary results extracted from the same execution scenario. The development was focused on

creating a Web-based application, using Struts (Struts 2010), running on JBoss Application Server (JAS) (JBoss 2010), and accessing a Relational Database through Hibernate API (Hibernate 2010). System modelling was done using an UML tool, and for Java development we used the Eclipse IDE (Eclipse 2010). The system was a logistic support analysis system called SISALI. At a certain point an update on the "Create Service Request" requirement was requested by the client. Such update was a 'cosmetic' change that, albeit simple, would affect the development schedule. To justify our denial to proceed with that change at that time, we decided to use the tool to show the user the amount of work that would be necessary to do it.

At that time, we knew two things: (1) from statistical analysis made in previous projects the average time to estimate impact change in a single functional requirement was 3 workdays (1 work day being equal to 8 hours) and used up the time of a senior analyst (1 senior analyst hour costs US$50.00); and (2) that it was difficult to make the client visualize the impact properly, in order to convince the client that the change should be postponed, as it would otherwise affect the project schedule (and cost). At this time, technical information is not the best way to get to the client, and it is more effective seeing the actual impact throughout the code graphically displayed.

By using this approach, and guaranteeing that all steps previously presented were followed, one can estimate impact change much faster, clearly reducing the overall cost, as the professional assigned to the analysis would not be unavailable for too long. In this example, the impact analysis was done in less than four hours and followed three steps as explained:

(1) Recompiling the software code;

(2) Generating the impact graph;

(3) Extracting and formatting related reports and presenting them to the stakeholders.

From the estimated 3 workdays, meaning a total 24 hours of conventional analysis to assess impact and a total cost of US$1,200.00, to only 4 hours using the MDA4Eclipse analysis module, we experienced a 20-hour reduction of expensive human resources, meaning an exact 83.3% saving in time and a US$1,000.00 cost cut. It also helped showing the client that, despite the simplicity of the change, it was unnecessary and would affect too much code, risking delaying project delivery.

Although this is a preliminary case study, the technique presented worked as expected and improved the way stakeholders visualized change impact on the system. This approach certainly helped us avoid spending effort with time-consuming manual analysis and unnecessary (re)work, as all client change requests were evaluated and concluded to be critical or non critical. It is certainly an interesting path to follow and encourages us to pursue further with this research. Finally we had a clearer view of how to speed up development time, saving man-hour costs, avoiding unnecessary (re)work that would be wasted with client needs that were not in fact critical to system behaviour.

Conclusions and Future Work

This paper presented preliminary results of a MDD change impact analysis approach. Our purpose was to improve the way requirement updates were evaluated and actually conducted in our projects. To support such action, we built a change impact tool in the MDA4Eclipse assessment module, capable of acquiring source code information, relating it to the functional requirements of projects, and creating a visual change impact graph.

By using this approach we were able to significantly reduce development time by means of severely reducing unnecessary (re)work and resource allocation; as further research work we intend to refine the environment to maximize its communication skills to produce better views and analysis results.

In spite of its being a preliminary analysis we can foresee more benefits to come, such as risk impact management. We also intend, as soon as we can, to improve the way we estimate the impact level simply from direct and indirect to a more meaningful representation such as a numeric impact factor (1 to 10). Such impact factor might come from using software metrics like fan-in and fan-out, depth of inheritance tree, lack of cohesion in methods, and others.

Finally, we look forward to creating a full-fledged environment where we can control the whole application development process, from the early stages of domain modelling, going through the code generation stage, enabling a management environment where stakeholders can benefit from several analysis mechanisms such as the one presented in this paper.

References

ArgoUML, UML Modelling Tool. http://argouml.tigris.org. Last access: April 2010.

Bohner A. and Arnold, R. S., Software Change Impact Analysis. IEEE Computer Society Press. Los Alamitos, California, 1996.

Bou, B., Treebolic Project, http://treebolic.sourceforge.net, last access: December 2009.

Eclipse Project, IBM, http://www.eclipse.org, last access: August 2010.

Frankel, D. F., Model Driven Architecture – Applying MDA to Enterprise Computing, Wiley Publishing, 2003.

Hibernate Project, Red Hat, http://www.hibernate.org, last access: August 2010.

JBoss Application Server, Red Hat, http://www.jboss.org/, Last access: August 2010.

Lamping, J., Rao, R., Pirolli, P., A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies. Conference on Human Factors in Computing Systems. Xerox Palo Alto Research Center, Palo Alto, CA, CHI 1995.

Law, J. and Rothermel, G. Whole Program Path-Based Dynamic Impact Analysis, Proceedings of the 25th International Conference on Software Engineering (ICSE'03), IEEE 2003.

Marzullo, F.P., Souza, J.M. and Xexéo, G.B., A Domain-Driven Approach for Enterprise Development, using BPM, MDA, SOA and Web Services, Journal of Eletronics and Computer Science, v. 10, Korea, 2008, pp. 1-6.

Marzullo, F.P., Souza, J.M., A Software Development Time Estimation Approach using BPM, MDA, SOA and Web Services, WorldComp 09, Las Vegas, NV, 2009.

Masticola, S. P., A Simple Estimate Cost of Software Project Failures and the Breakeven Effectiveness of Project

MDA4eclipse Research Project, www.mda4eclipse.com.br. Last access: April 2010.

MindTools, Impact Analysis, http://www.mindtools.com/pages/article/newTED_96.htm. Last access: August 2010.

Model Driven Architecture, http://www.omg.org/mda. Last access: December 2009.

MOF 2.0/XMI Mapping Specifications, v 2.1.1, http://www.omg.org/technology/documents/formal/xmi.htm. Last access: December 2009.

Pressman, R. S., Software Engineering - A Practitioners Approach, McGrawHill , 2005.

Project Management, Institute. A Guide to the Project Management Body of Knowledge, PMI, 2004.

Rajlich, V. and Gosavi, P., Incremental Change in Object-Oriented Programming, IEEE Software, 2004.

Risk Management, First International Workshop on the Economics of Software and Computation (ESC'07), IEEE 2007.

Rodrigues, G. N., A Model Driven Approach for Software System Reliability, Proceedings of the 26th International Conference on Software Engineering (ICSE'04), IEEE 2004.

Struts Project, Apache, http://struts.apache.org/, Last access: August 2010.

Sun Microsystems Corporation, Annotations, http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html, 2010.

Sundman J., Impact Analysis with Rational Architecture Management Tools, IBM, Software Group, 2007.

Zhang S., Gu Z., Lin Y., Zhao, J., Celadon: A Change Impact Analysis Tool for Aspect-Oriented Programs, ICSE'08, Leipzig, Germany, 2008.