Business Requirement Negotiation based on Generalized Requirement Approach (GRA)

Aleksandar Bulajic*
* LANB, Kongens Lyngby, Copenhagen, Denmark
LANB@45.dk

***Abstract*—Business software development is based on the specific business requirements that are collected during requirement negotiation process. Gathering business requirements, when final product requirements are dictated by known client, can be a difficult process. An idea about new business product can be obscure, and described by general terms that contributes very much common misunderstandings. Business requirement verification accomplished by using text and graphics, and manual review processes, can be slow, error prone and expensive. Misunderstandings and omitted requirements affect future software product. This research work proposes new approach to requirement negotiation, the Generalized Requirement Approach (GRA) and is focused on demonstration of business requirement during requirement negotiation process. The process of the business requirement negotiation is guided by the set of predefined objects that store requirement description in the common repository, in the structured text format. The objects attributes and properties are guidelines for specifying sufficient level of requirement details for generating source code that is used for requirement demonstration. The source code and executables are generated without manual programming.**

I. INTRODUCTION

Business requirement specification is one of the most important documents for software development project. The contract signing, budget, time scheduling and resource allocation depends hardly on the correct business requirement specification. Omitted and misunderstood requirement can cause huge revision and code refactoring in late software development phases and affect project budget and duration.

Gathering business requirements, when final product requirements are dictated by known client, can be a difficult process. An idea about new business product can be obscure, and described by general terms that contributes very much common misunderstandings. Business requirement verification accomplished by using text and graphics, and manual review processes, can be slow, error prone and expensive.

Research studies show that issues related to requirements that are discovered in later project phases produce even greater costs and delays. Discovering or modifying requirements in the Design Phase could be three to six times more expensive. In the Coding Phase it is up to 10 times more expensive, and in the Development Testing Phase it is 15 to 40 times more expensive. In the Acceptance Phase, it is 30 to 70 times more expensive, and in Operation Phase it could be 40 to 1000 times more expensive. [1]

The IBM Project Management presentation use the Meta Group study to illustrate that 70% of large IT projects failed or did not meet customer expectation. [2]

This research work proposes new method for business requirement negotiation process called Generalized Requirement Approach (GRA). The GRA requires demonstration of business requirements during requirement negotiation process. To be able to demonstrate requirement, the GRA requires the GRA Framework. The GRA Framework is implementation of the GRA method. The GRA method is described in the "Generalized Requirement Approach (GRA)" section. The GRA Framework is described in the "GRA Framework (GRAF)" section.

The GRAF guides process of the business requirement negotiation by a set of predefined objects that store requirement description in the structured text format in the common repository. The object attributes and properties are guidelines for specifying sufficient level of requirement details for generating source code [3]. Automated build is using source code to create executables and demonstrate requirement on fly. The source code and executables are generated automatically without manual coding by the Generic Programming Units (GPU). The GPU is a class or module responsible for generating source code. The GPU is based on the parameterized methods. The GPU is setting method parameters to the values stored in the structured text format before generating source code. Besides changing parameters and generating methods, the GPU is able to generate user interface, classes, SQL statements and configuration files. The GPU is described in the "GRA Framework (GRAF)" section.

The GRA addresses requirement management syndromes, specification of Insufficient Details Level [3], the IKIWISI (I'll know it when I see it) syndrome, the "Yes, but' syndrome ('that is not exactly what I mean') and the 'Undiscovered Ruin' syndrome ('Now that I see it, I have another requirement to add').

II. RELATED WORK

Traditional requirement management approach is often identified by the Waterfall [4][5] software development method, where comprehensive requirement analysis and documenting is completed before a start of the next project phases. On the contrary, the Agile Requirement Management [6] does not wait that all requirements are specified, neither is waiting that a whole requirement is specified. A development starts as soon as a part of the requirement is understood [7]. The project is developed by using an iterative and incremental approach. The Agile software development process is based on the short

development iterations. Each of iteration implements a limited number of requirements. The next iteration is planned on the user feedback and experience collected during iteration testing process [7]. Short iteration advantage is early discovering of the requirement misunderstanding. However, if requirement is misunderstood, then time spend for code development can be waste and affects project scheduling and budget. Requirements that are implemented in the next iteration can require code refactoring. Huge code refactoring can affect project budget and scheduling.

Mc Connell [8] pointed to importance of software project proper preparation and prerequisites such us planning, requirements, architecture and design.

The Test Driven Development (TDD) is Extreme programming method based on the test first approach. The test is created before implementation code [9]. The TDD improved test coverage and promotes testing culture [10]. While low test coverage can mean that test was not properly executed, high test coverage guarantee nothing [11].

The Microsoft Solutions Framework (MSF) is Microsoft best practice method for delivering software according to specification, on time and on budget. [12] "The MSF philosophy holds that there is no single structure or process that optimally applies to the requirements and environments for all projects. It recognizes that, nonetheless, the need for guidance exists." [12]

Hewlett-Packard experimented by implementation of the Evolutionary Development method (EVO) "to improve software development process, reduce number of late changes in the user interface and reduce number of defects found during system testing" [13]. The first and second attempt that used two weeks delivery cycles and four to six delivery cycles over more than year and half failed to delivery expected features and expected results. [13] The third attempt that used first month to prototype after 4,6 months of implementation delivered world class product. [13]. These experiments on the full scale industrial projects confirmed importance of prototyping as a tool for requirement clarification.

The Unified Software Development Process, an iterative and incremental component based software development method that is case driven, architecture centric and risk focused has been created in 1999. [14]

Road map in the Unified Process method is described as The problem domain, Stakeholder needs, Moving Toward the Solution Domain, Features of the System, Software requirements.[14] A Problem Domain is identified by Needs, while Features and Software Requirements belong to the Solution Domain.[14]. The most known implementation of the Unified Process (UP) is IBM Rational Unified Process (RUP) component-based process.

However, the first step in the software development process is requirement description and clarification. Collecting and describing requirements in the Requirement Specification document can be a difficult job. The natural language is subject of different interpretation and cause ambiguities.

The IEEE 1998b standard describes characteristics of a good requirement specification such as correct, unambiguous, complete, consistent, traceable, verifiable [15].

The Unified Approach [14] added to this list next characteristic "understandable".

Other authors, such as Wiegers, describe characteristics of excellent requirement by requirement statements characteristics such as complete, correct, feasible, unambiguous, and verifiable [16].

Wiegers makes differences between Requirement Description and Requirement Specification description and a good Requirement Specification describes as complete, consistent, modifiable and traceable [16].

Requirements verification is a process of improving requirement specification according to recommendation of good requirement description practice.

Wiegers [16] favor technique for requirement verification is formal inspection of requirements document accomplished inside of the small teams where are represented different views, such as analyst view, customer view, developer and tester view. This technique is supported by testing requirements by developing functional test cases and specifying acceptance criteria [16].

Rational Unified Process [14] use traceability matrix for requirement verification. A requirement or a need in RUP terminology is linked to a feature. A Feature is linked to a Software requirement and Use Case. Use Case is linked to Test Cases. If some of the links is missing it is considered an indication that requirement is not properly verified. Requirement verification in this case is considered done if a link to a Use Case and a Test Case exists [14].

Sommerville [17] for requirement verification process specifies requirement reviews, test case generation and automated consistency analysis in case when requirements are specified "as a system model or formal notation".

Prototyping technique is used for requirement validation. Sommerville see prototyping as a requirement verification technique [17]

Requirement validation is a process of "evaluating of software component during or at the end of development process" [18]

Prototyping is an effective method for requirements clarification, proof of concept and reducing a risk that final product is significantly different than expected [16].

Requirement verification accomplished by using text and graphics, and manual review processes, can be slow, error prone and expensive.

Omitted and misunderstood requirements can cause huge revision and code refactoring in late software development phases and affect project budget and duration.

III. SOFTWARE DEVELOPMENT METHODOLOGY (SDM)

The Software Development methodology is software development process that can be described by following development phases and activities:

- Analysis – system requirements management,
- Architecture & Design –system design,
- Development – internal design and coding ,
- Test – test and validation,
- Deployment – operation and maintenance.

The SDM is a structured approach to software development. The SDM purpose is production of high-quality software in a cost-effective way [17]. The structuring process purpose is to enable process planning and controlling. The SDM process structure is implemented in the different software methodologies, sequential and iterative, incremental and evolutionary, rapid application development and prototyping

History of the Software Development Methodology (SDM) started in the 1956 when Herbert D. Benington presented his paper "Production of Large Computers Programs" at "Symposium on advanced programming methods for digital computers: Washington, D.C., June 28, 29, 1956" by [19].

Dr. Winston W. Royce in 1970 presented his personal view about managing large software developments in his paper "Managing the Development of Large Software Systems" at "Proceedings of IEEE WESCON 26 " [20]. While Herbert D. Benington called the first phase, where broad requirements are defined, the Operational Plan phase, Dr. Winston W. Royce called the first software development phase the System Requirements phase.

The process of the requirement specification, verification and validation is described in the Figure 1 "Traditional Requirement Management Approach":

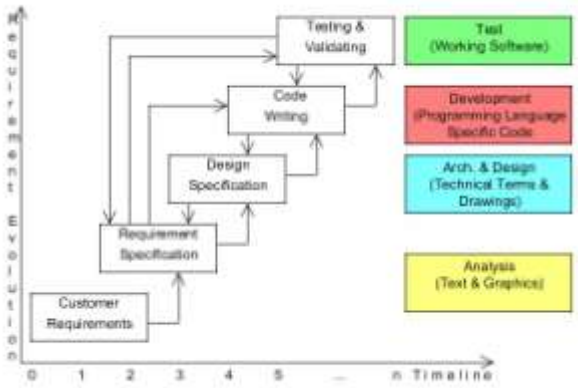

Figure 1 "Traditional Requirement Management Approach"

The requirement verification is understood as a process of the initial requirement evaluation, executed during requirement gathering, elicitation and specification. [18] The requirement validation is understood as a process of the requirement evaluation after completing of the development phase. [18]

The output from the Traditional Requirement Management is Requirement Specification document. The Requirement Specification document is used as reference document for further software development planning's' and activities, Design Specification, Code Writing and Testing & Validating, even it is well known that a written texts as well as graphics are ambiguous and subject of different interpretations.

The choice of software development method affects time distance between requirement specification and requirement validation. In case of the Agile development methods this time distance can be a week or weeks long. In case of more traditional approaches, this can be a month or months long.

Traditional requirement management, Waterfall like method, is most appropriate for a project where requirements are stable and do not change during software development process. However, analysis shows that an average of 25 % of requirements change in the typical project, and change rate can go even higher to 35% to 50% for large projects [21].

If time difference between requirement specification and requirement validation is longer, then is most likely that requirement will be changed.

This process can be improved by introducing requirement demonstration as early as possible to avoid waste of time and resources on implementation and modification of misunderstood requirements.

IV. Generalized Requirement Approach (GRA)

The Generalized Requirement Approach (GRA) solution proposes requirement validation prior to creating Requirement Specification. Requirement validation requires creating of the executables that are created from the source code. Writing source code manually can be slow and error prone process.

The GRA method proposes automatic source code generation from structured textual descriptions that are expressed by customer native language. The process of describing requirements, generating source code and requirement demonstration is called Requirement Normalization process. The Figure 2 "Generalized Requirement Approach Overview" illustrates proposed solution:

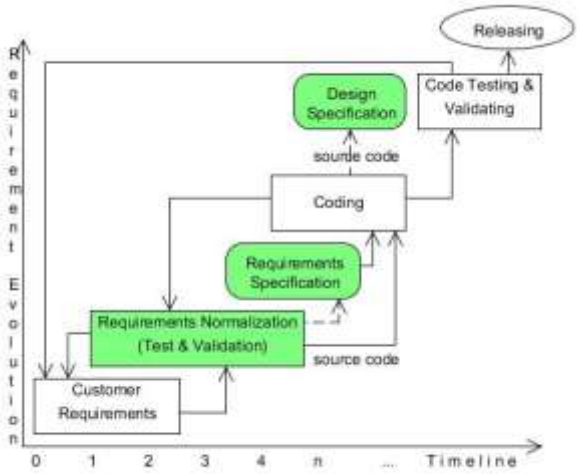

Figure 2 "Generalized Requirement Approach (GRA) Overview"

The Requirement Normalization process is responsible for:

- Guiding user to specify sufficient level of details [3] by using customer native language,
- Storing requirement description in the structured text format,
- Automatic source code and executables generation.

Besides requirement description, the Requirement Normalization primary goal is to clarify obscure customer's requirements. The Requirement Normalization process is considered complete when requirement is possible to describe by sufficient level of details from which is possible generate source code and build executables. The outputs from the Requirement Normalization are generated Requirement Specification and source code. The source code can be used in the next project phases.

While traditional requirement management writes Requirement Specification document, the Requirement Specification in case of the GRA method is stored in the central repository and can be generated on demand. It is not recommended direct update of the Requirement Specification. Updates should be accomplished though Requirement Normalization process.

Based on discussion in this section are identified following GRA features:

- Document and store requirements in the structured text format described by customer native language,
- Generate source code without manual programming,
- Demonstrate working software during requirement negotiations process,

V. Generalized Requirement Approach Framework (GRAF) Overview

The Generalized Requirement Approach Framework (GRAF) is implementation of the Generalized Requirement Approach (GRA) method. The GRAF contains code, classes, objects and libraries that are guiding user during requirement negotiation process to provide detailed requirements specification that is sufficient to generate source code and executables. The GRAF is responsible for implementation of the GRA features.

The Figure 3 "The Generalized Requirement Approach Framework Design" illustrates the GRA framework high level design:

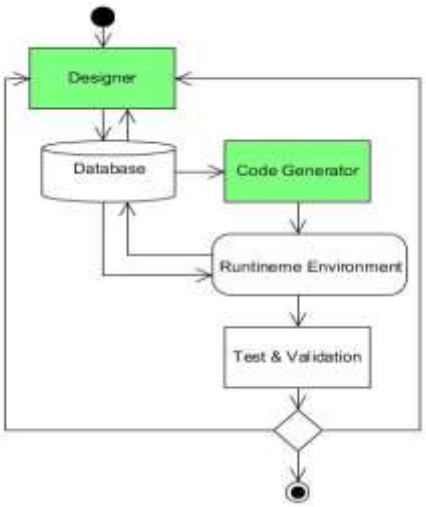

Figure 3 "The GRA Framework Design"

The GRAF is organized around central repository. In the central repository are stored requirement descriptions and used by Code Generator when necessary.

Designer is responsible to store structured text format descriptions in the Database and for guiding a user to specify sufficient amount of details. Omitting sufficient number of details during requirement specification can affect project duration and increase overall cost expenses [3].

Code Generator is responsible for generating source code by using structured text data stored in the Database. The source code is generated in the standard programming language, for example C# or Java. The generated source code is executed in the Runtime Environment. The Runtime Environment depends of the generated source code. For example, if the C# source code is generated by Code Generator the Runtime Environment needs Microsoft .NET and CLR installation. If the Java source code is generated by Code Generator the Runtime Environment needs JRE installation.

The Test & Validation process validate requirements by using code that is executed in the Runtime Environment. If requirement does not satisfy expectations the process can be repeated and retuned back to Designer.

The GRA method can be implemented by using different technologies, such as Microsoft .NET, Java or JavaScript. In this paper the GRA Framework is implemented by using the Microsoft .NET and C# language. Each implementation can be based on the different object types.

The GRA Framework used in this paper identifies following groups of objects that are used by Designer during requirement negotiation process:

- Objects responsible for requirement description and documenting such as Requirement, User Story, Use Case, and Test Case.,
- Objects responsible for storing data in structured text format that are used to generate source code such as Forms, Data Sources, Application Objects and Interfaces.

Each of the GRA Framework object is mapped to one or more corresponding database entities that are used for storing data in the structured text format and for retrieving data when the GRA Framework need it.

Objects responsible for requirement description and documenting are designed according to best practice [22].

Objects responsible for storing data in structured text format are business application building blocks and in this particular GRAF implementation are used following objects:

- Form object is describes entry fields and other predefined User Interface (GUI) controls that enable user and software application interactions,
- Data Source object is responsible for creating database tables and relations,
- Application Object is responsible for backend and batch job processing,

- Interface object is in the same time Application Objects, but for this kind of objects is specific communication with sources of data external to application.

Objects responsible for storing data in structured text format are used to generate source code. Code Generator designed for this paper is illustrated in the Figure 4 "The GRA Framework Source Code Generation":

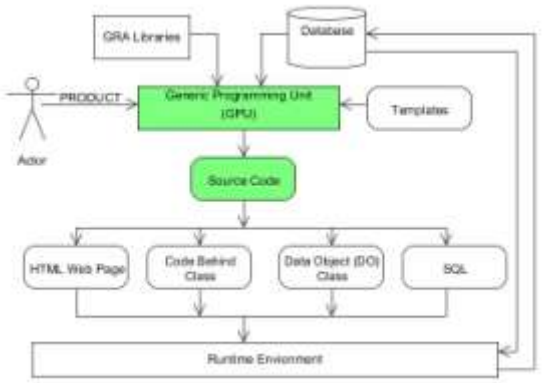

Figure 4 "The GRA Framework Source Code Generation"

The source code is generated from the structured text descriptions, the GRA Libraries and Templates. The structured descriptions are stored in the Database tables. The GRA Libraries are containing parameterized methods and templates. These methods and templates are adapted to requirement specifics, and inserted in the generated source code. The Templates contains controls and controls attributes that are specific to implementation technology. For example if it is generated ASP.NET source code, the Templates are adapted to the ASP.NET controls such as Textbox, Button or Dropdown list, as well as to the ASP.NET language specific syntax. The methods and templates are used as building blocks to create source code. The process of source code generation is initiated externally from Actor by sending a name of the object that needs to be generated.

The Generic Programming Unit (GPU) is a code, a method, a class or a module that is able to generate source code. One example of the GPU is a GPU that can generate form. The form can be described by form's name, field name, field type, data type, data length and number of decimal places, and form's control type, such as text field, drop down list, check box, button, etc. The GPU from this data shall be able to create form that can insert, modify and delete entries, and execute action code that is assigned to the form fields.

The Generic Programming Unit (GPU) is glue that connects database structured text descriptions, library methods and templates, and creates source code. The GPU is reading data stored in the Database for each particular object and creates source code according to requirement description by using GRA Libraries and Templates. The GRA Libraries contains templates and methods. The outputs from the GPU are HTML Web Page, Code Behind Class, Data Object Class and SQL. The Data Object Class is responsible for data mapping from relational database to objects and is a part of the Data Access Object pattern implementation. The generated SQL statements are used in the implementation of CRUD database operations. The GPU shall be able to generate other source code if there are available sufficient details of information and if implementation technology can support it.

The Runtime Environment is responsible for execution of the generated source code and is using Database for storing and retrieving application data.

VI. EXPERIMENT

The GRA Framework implementation is tested on the Retail Store application. The Retail Store is a fictive E-Commerce application described by following Retail Store User Story:

"As the Retail Store we want to sell our products on-line through Internet in order to increase product availability, get in touch with more customers and increase sale and profit". From the Retail Store User Story is possible to identify:

- ProductComponent object,
- Sales Operation.

The Product Component requirements are described in the Salesman User Story as "a *need to add, update and remove product from the product list"*. The Sales Operation requirements are further elaborate in the Buyer User Story as "a *need to select product, add product to shopping cart, create order and enable online payment by credit card"*.

The source code is generated according to the process described in the Figure 4 "The GRA Framework Source Code Generation generated source code. The source code has been generated from description of the Product, Shopping Cart, Order and Payment forms. Each form is described by forms name, field name, field type, data type, data length and number of decimal places, and forms control type, such as text field, drop down list, check box, button, etc. The GPU from this data generates forms that can insert, modify and delete entries, and execute action code that is assigned to the form fields. The Figure 5 "Product Form" illustrates generated Product form:

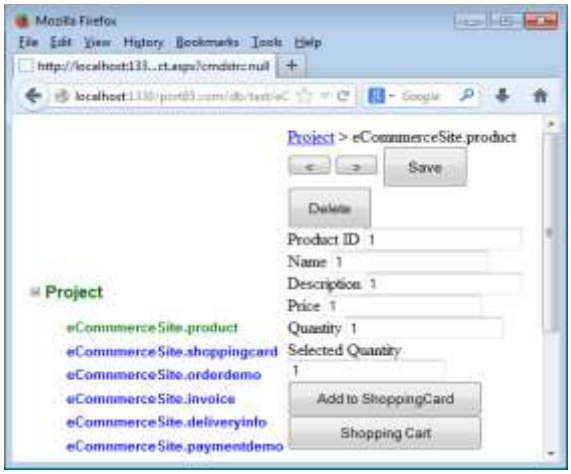

Figure 5 "Product Form"

Other forms are generated the same way and the GPU generates the fully functional application that is able to demonstrate Product and Sales Operation components. The user of generated application is able to enter, store,

update and browse data, add data to Shopping Cart, change selected quantity and review Order before executing payment operations. To the Order form are assigned calculations for calculating items price, handling fee and VAT amount, and for calculating Order total amount. The limited space in this paper does not allow full presentation of the generated application.

VII. CONCLUSION

The result of the experiment, E-Commerce fictive application, demonstrates feasibility of proposed solution, and shows that the predefined set of the framework objects and code that uses data defined during requirement negotiation process is sufficient to generate source code and generate application without a need for writing code manually.

The Generalized Requirement Approach (GRA) proposed in this paper can improve software development productivity, and improve the quality of the final product.

However, effective use of the GRA method requires implementation of the GRA Framework (GRAF). The GRAF objects attributes are guidelines for specifying requirement and providing sufficient details level. While in the existing Software Development Methodology source code is written by programmers manually, the GRAF generates source code from the requirement user descriptions stored in the central repository.

The proposed solution can contribute to:

- Clarify requirements and improve requirement understandings,
- Address IKIWISI, "Yes, but", "Undiscovered Ruin" and "Insufficient Details Level" requirement syndromes,
- Closing a gap between requirement specification and requirement validation,
- Producing of an environment where requirements can be executed, analyzed, observed, and validated,
- Promote customer active participation.

The generated source code and executables are fully functional application that can be executed and tested. The Retail Store demo application can demonstrate workflow, data, algorithms, and can be used for ad-hoc testing.

According to the currently collected experience, the critical part of this approach is providing sufficient amount of the features that are in the GRAF represented by Application Object. Application Object represents classes and generic methods that solve particular programming issue. For example it can be testing of unique Id, moving rows from one relation table to other or creating new entities that are combination of the existing entities. In the Retail Store demo application example such example is the addRowToDataSource generic method. The addRowToDataSource method is able to add current data source row to any other data source. In this GRAF implementation, the target data source is specified during requirement negotiation and is stored in the requirement description.

This framework version is developed for research and experiment purposes. The further development can create a product that besides requirement negotiation can be used for estimation, and generally speaking for project management purposes

REFERENCES

[1] DragonPoint, Inc (2008), Company Newsletter issue No. 3, "Requirements Capture: Keys 6 Through 10 to a Successful Software Development Project", available at http://www.dragonpoint.com/CompanyNewsletters/RequirementsCaptureKeys610.aspx

[2] IBM (2007), "IBM Project Management", available at http://facweb.cs.depaul.edu/yele/Course/IS372/Guest/Dawn%20Goulbourn/IBM%20PM%20presentation%20for%20DePaul.ppt

[3] Bulajic, Aleksandar, Stojic, Radoslav, Sambasivam, Samuel (2013), "Gap Between Service Requestor and Service Provider", "Applied Internet and Information Technologies" ICAIIT2013 , Zrenjanin, Serbia, October 26, 2013

[4] Benington, Herbert D. (1956), "Production of Large Computers Programs", Symposium on Advanced Programming Methods for Digital Computers sponsored by the navy Mathematical Computing Advisory Panel and the Office of Naval Research, June 1956,

[5] Royce, Winston W. (1970), "Managing The Development of Large Software Systems", Proceedings of IEEE WESCON 26, August 1970,

[6] Beck Kent, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas (2001), "Manifesto for Agile Software Development", available at Internet http://agilemanifesto.org/

[7] Beck, Kent (2002), " Introduktion til Extreme programming", IDG, 30-05-2002

[8] Mc Connell, Steve, "Code Complete 2: A Practical Handbook of Software Construction", Microsoft Press, 2004

[9] Beck, Kent (2002a), "Test Driven Development by Example", Addison-Wesley , November 18, 2002

[10] Bulajic, Aleksandar, Sambasivam, Samuel, Stojic, Radoslav (2012), "Overview of the Test Driven Development Research Projects and Experiments", Informing Science and Information Technology Education 2012 Conference (InSITE) in Montreal, Canada, June 22-27, 2012

[11] Cornett, S. (2011). "Minimum acceptable code coverage", Bullseye testing Technology, 2006-2011

[12] "Microsoft Solution Framework 3.0 Overview" (2003), "Microsoft Solution Framework White Paper", Microsoft, 2003

[13] May, Elaine L., Zimmer, Barbara A. (1996), "The Evolutionary Development Model For Software", Hewlett-Packard Journal, August 1996

[14] Leffingwell, Dean, Widrig, Don (2000), "Managing Software Requirements : A Unified Approach", Addison-Wesley,2000

[15] "IEEE Recommended Practice for Software Requirements Specification" (1998), Software Engineering Standards Committee of the IEEE Computer Society

[16] Wiegers, Karl E. (2003), "Software Requirements", Microsoft Press, A Division of Microsoft Corporation, One Microsoft Way, Redmond, Washington, 2003

[17] Sommerville, Ian (2001), Software Engineering 6th Edition", Pearson Education Limited

[18] "IEEE Standard Glossary of Software Engineering Terminology" (1990), IEEE-Std 610.12, IEEE Standard Board, September 28, 1990

[19] Benington, Herbert D. (1956), "Production of Large Computers Programs", Symposium on Advanced Programming Methods for Digital Computers sponsored by the navy Mathematical Computing Advisory Panel and the Office of Naval Research, June 1956,

[20] Royce, Winston W. (1970), "Managing The Development of Large Software Systems", Proceedings of IEEE WESCON 26, August 1970

[21] Larman, Craig (2005), "Applying UML and Patterns", Pearson Education, 2005

[22] Cockburn, Alistair(2001), "Writing Effective Use Cases", Addison-Wesley