THE PROJECTIT-STUDIO/REQUIREMENTS CASE TOOL
A Practical Requirements Specification Case Study

David Ferreira[1], Alberto Rodrigues da Silva[1]

david.ferreira@inesc-id.pt, alberto.silva@acm.org

[1] INESC-ID, Instituto Superior Técnico, Rua Alves Redol 9, Lisbon, Portugal

Abstract: Adopting a pragmatic attitude, regarding to the high rate of unsuccess verified among IT projects, and also bearing in mind the importance of the software development process's early stages to overcome this problem, we embraced the goal of creating a CASE tool for requirements specification and validation. Our approach is based on a controlled natural language designated as ProjectIT-RSL and seeks to achieve a higher level of rigor and quality of the produced requirements specifications. The supporting tool, the ProjectIT-Studio/Requirements plugin, belongs to a broader workbench called ProjectIT-Studio, whose main goal is to cover the entire software development life-cycle, providing guidance during the process according to software engineering best practices and methods. This paper presents a small but illustrative requirements specification case study, which emphasizes the current tool's support and its inherent usage advantages during the requirements specification process.

Keywords: Requirements. Specification. CASE tool. Controlled NL and NLP.

1. Introduction

Despite the resemblance with the conventional engineering areas and the efforts made during the last few decades, the software development process is still an immature activity, presenting several critical issues that compromise its success, namely in terms of scheduling, budget, and quality of the software product delivered. The main cause to this situation is the non adoption of a standard and systematic approach founded on the best practices and methods proclaimed by the software engineering community (Dutta et al., 1999). Independently of the degree of formality adopted, requirements in their essence describe what the system should do or at least a constraint on its behavior (Sommerville and Kontonya, 1998), which is obviously critical for the success of the whole development process. Several surveys and studies (Standish Group International, 1994) have emphasized the costs and quality problems that can result from mistakes in the early phases of system development, such as inadequate, inconsistent, incomplete, or ambiguous

requirements (Bell and Thayer, 1976). Thus, bearing in mind that the early detection of errors in the upstream phases of the product's life cycle allows minimizing their cost and effort impacts (Cybulski, 2001), it's straightforward to comprehend the growing importance of requirements engineering.

Therefore, our proposal seeks to reduce these negative aspects of the software development process by introducing a controlled natural language for requirements specification, supported by a CASE tool which offers validation mechanisms. This language, ProjectIT-RSL, is based on the identification of common linguistic patterns of requirements documents and it allows to significantly surpass one of the greatest challenges of requirements engineering, namely to minimize the gap between the stakeholder's vision and the model captured by the requirements engineer. By using a natural language approach, non-technical users can easily specify, in a semi-formal manner, the intended requirements. The ProjectIT-Studio/Requirements is a specialized CASE tool whose underlying parsing mechanisms provide the on-the-fly validation required to determine the specification's consistency in terms of syntactic and semantic rules, hence allowing a deep integration with MDE tools (Schmidt, 2006).

This paper describes our CASE tool and highlights the inherent potential benefits of its application in real-world projects, in terms of productivity and quality, when integrated with the remaining tools of ProjectIT-Studio workbench. Section 2 overviews the ProjectIT initiative and its CASE tool. A detailed tool analysis of the ProjectIT-Studio/Requirements is provided in Section 3. Section 4 presents an illustrative case study to emphasize the practical application of the ProjectIT approach concerning the requirements engineering phase, based with this academic exercise. Finally, Sections 5 and 6 present, respectively, related work and conclusions about of our research based on this experience, which justify our perception that this proposal provides innovative contributions.

2. ProjectIT Overview

In our research we consider that the emphasis in software development projects should be stressed in the project management, requirements engineering and design activities, and consequently the effort in production activities, like software programming and testing, should be minimized and performed as automatically as possible. Therefore, to substantiate this vision, we started an initiative in the area of requirements engineering and model-driven development, which we have named ProjectIT (Silva, 2004). Currently, the ProjectIT approach is focused upon interactive systems, which are a subclass of information systems driven by human-machine interaction and managed through role-based access control.

One of the results of this initiative is ProjectIT-Studio, which is a modular and extensible workbench that we have developed to support and test our research ideas (Silva et al., 2006). It provides an integrated environment that supports some of the most important tasks of the software development life cycle, such as requirements specification, architecture definition, system design and modeling, and code development. In addition, and because it promotes productivity, ProjectIT-Studio provides innovative features such as: (1) requirements-to-models, models-to-models, models-to-code transformation techniques; (2) template managing; and (3) UML profile definition. This tool consists of an orchestration of plugins for Eclipse.NET (illustrated in Figure 1), which are integrated via Eclipse.NET's plugin communication mechanisms. Each one of the three main plugins of ProjectIT-Studio, easily identifiable in the upper layer of Figure 1, is further described in (Silva et al., 2006).

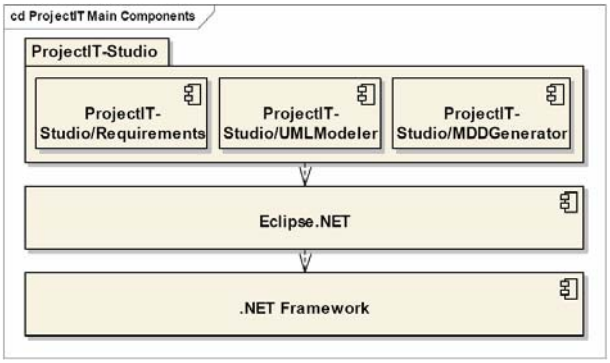

Figure 1 – ProjectIT-Studio main components

3. Related Work

The use of natural language in the initial phases of the software development process has received attention for more than 20 years. Therefore, ProjectIT-Requirements shares several similarities with other requirements specification languages research projects. This work attempts to incorporate the best concepts and techniques available, namely from: (1) CIRCE (Ambriola and Gervasi, 2003), which proposes a "lightweight formal method" approach based on fuzzy matching domain-based parsing techniques to produce a formal validation of requirements written in natural language, complemented with a multiperspective-based GUI approach to provide feedback to the user; (2) ACE (Fuchs and Schwitter, 1996),

which presents the concept of using a controlled natural language to write precise specifications that enable their translation into first-order logic; and (3) NL-OOPS (Mich and Garigliano, 1999), which influenced this work by presenting its knowledge-base and inference-engine architecture. Moreover, it applied the approach proposed by Abbot (Abbot, 1983) of a syntax highlighting mechanism, emphasizing the suggested implicit relations between the following natural language morphologic categories and the respective computational concepts pairs: nouns and classes/entities; adjectives to describe attributes, and verbs to identify methods. Besides these research projects there is another class of tools, comprising well-know commercial applications such as RequisitePro (available at www.ibm.com/software/awdtools/reqpro/) and DOORS (available at www.telelogic.com/doors/). These Requirements Management Tools offer deeply integrated solutions with a strong emphasis on traceability issues and usability, an issue typically disregarded by research projects due to their specificity. ProjectIT-Studio/Requirements stands in the middle because, although it presents a strong focus on its integration with ProjectIT-MDD component tools, it additionally offers NLP features like other research projects. This ability allows overcoming the critical limitation of state of the art commercial tools of treating requirements as black-boxes, i.e., ignoring their meaning.

4. ProjectIT-Requirements

ProjectIT-Requirements (Videira and Silva, 2004) is the component of the ProjectIT architecture that deals with requirements issues. Its main goal is to develop a model for requirements specification, which, by raising their specification rigor, facilitates the reuse and integration with development environments driven by models.

4.1. The Specification Language

The ProjectIT-RSL is defined by following a simple approach, which started with the analysis of the format and structure of requirements documents arising from projects we have developed in the past few years. Then, we identified a common set of linguistic patterns for the requirements of this type of systems. From these patterns, we determined not only the main concepts used in requirements specifications, but also how they are structured and organized, including how they relate with each other and how they are combined into wider scope blocks. We derived a metamodel of the identified concepts, which is aligned with the XIS UML profile. However, the complete specification of all ProjectIT-RSL rules is beyond the scope of this paper (for more detail see (Videira et al., 2006a)).

4.1.1. Metamodel

Most of the sentences used in typical interactive systems requirements specifications indicate what the system should do, and present an operational perspective of the system. This is why they follow a simple and common pattern, in which a *subject* (the **Actor**, which is an active resource) performs an **Operation** (which can be atomic or not, usually expressed by a *verb*), which affects an *object* (the **Entity**, which is a static resource), eventually modifying its state through its **Properties**. Other more elaborated sentences can also include a condition, which describes a constraint about a simpler assertion. Another frequently occurring pattern specifies the attributes that define an entity; these are mainly declarative sentences, whereas the previous ones are imperative. The analysis of these patterns led us to the identification of the main concepts schema of our language, which are presented in Figure 2.

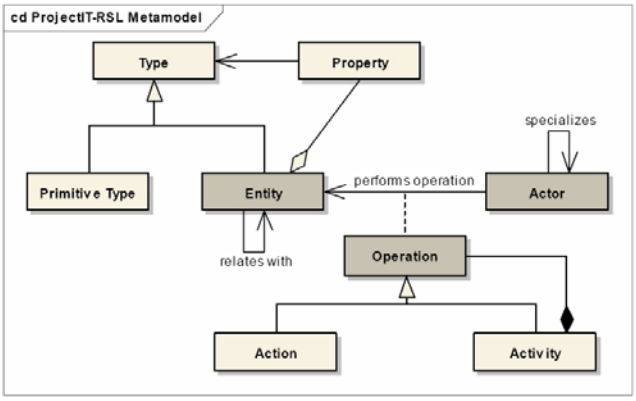

Figure 2 – ProjectIT-RSL metamodel

4.1.2. Structure Model

One of the best practices in requirements specification is the adoption of a standard structure in requirements documents, by following a predefined template and use cases writing patterns (Adolph et al., 2002; Cockburn, 2000). This not only eases their reading by different stakeholders, but also allows simplifying the implementation of different validation techniques. That is why our requirements documents are organized in specialized sections, which group together related sentences in terms of their goals. The achieved structural model, presented in Figure 3, clearly separates documentation from specification information. The first group comprehends introduction and comment sections which are useful for providing a synopsis and in loco specification rationales, respectively; whereas the second group contains more specific sections, being their names suggestive of their

intended purpose, namely grouping together business entities, functional requirements, and non-functional requirements specifications.

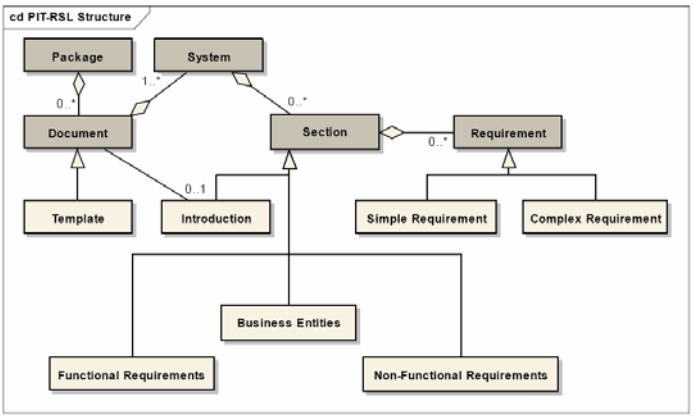

Figure 3 – ProjectIT-RSL structure model

4.1.3. Language Extensibility

The common linguist patterns, which are implicitly synthesized in the metamodel, are manually codified into Template Substitution (TS) rules (Videira et al., 2006a) that materialize the expressed concepts and relations of the ProjectIT-RSL language. A general purpose fuzzy-matching shallow parser (Videira et al., 2006b) uses these rules, consuming them along with the requirements document as input to extract knowledge from free-form controlled natural language requirements, grouped by the previously mentioned structure primitives. By using these TS rules to support the ProjectIT-RSL language we gain a great flexibility and an inherent extensibility mechanism since, depending on the problem domain of the system to build, we can modify our language accordingly. This per-project customization allows us to achieve a formalism comparable to the conventional concept of a Domain Specific Language (DSL) (Fowler, 2007). For further detail, the reader may consult (Videira et al., 2006a).

4.2. The CASE Tool

As depicted in Figure 1, the ProjectIT-Studio/Requirements tool is built upon Eclipse.NET framework. It encapsulates specific text editor's behavior and the base classes responsible for the plugin initialization, logging, and termination. It contains the ProjectIT-RSL Text Editor, shown in Figure 4. Additionally, it provides several tree-view components, and provides a coherent set of IDE features that extend the base platform with specific behavior to support the ProjectIT-RSL

requirements specification process. The Text Editor implements the vision to build a tool for writing requirements documents, like a word processor that detects and gives feedback from errors violating the grammar rules, as described in (Videira et al., 2006a). This component implements all the required features to assist the stakeholder during the requirements specification process. Namely, it provides syntax-highlighting, auto-complete, on-the-fly error checking with text annotation, and other kind of graphical annotations such as tasks and bookmarks, providing instant feedback on most of the syntactic and semantic problems, or avoids them entirely. Still, with the goal of keeping the tool compliant with OMG's standards, the tool provides the possibility to export to the XMI format, thus supporting the integration with other modeling and MDE third-party tools. For further information on the internal parsing mechanisms, and knowledge-base with inference-engine capabilities provided by the Jena .NET Port, the reader may consult (Videira et al., 2006b).

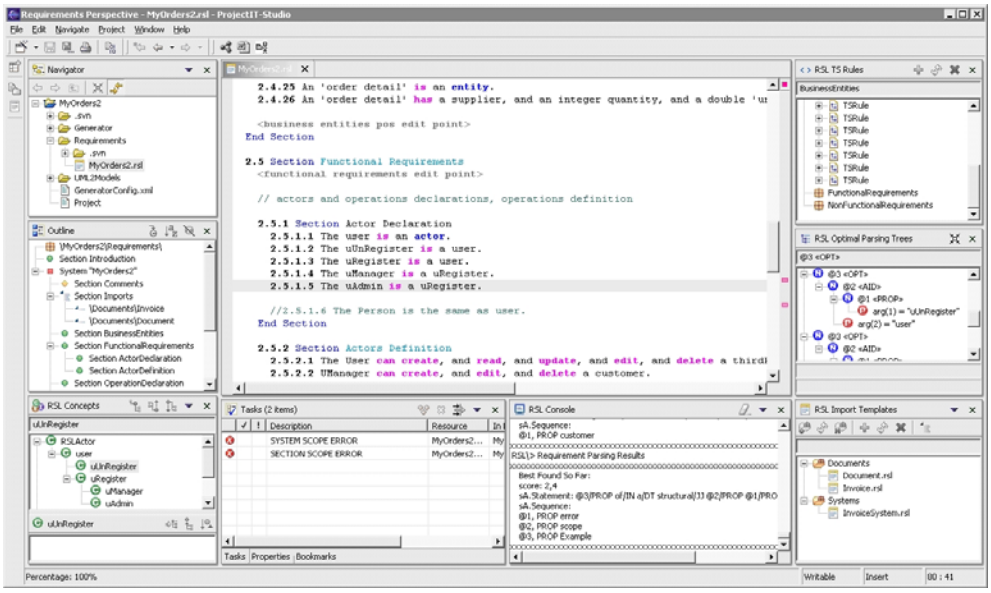

Figure 4: PIT-Studio/Requirements screenshot

5. MyOrders2 Case Study

This section presents a practical application of the concepts previously discussed in the paper, emphasizing the benefits of the ProjectIT approach and its support given by the ProjectIT-Studio/Requirements tool. This discussion follows a simple case

study, named MyOrders2, which is a simple management information system, specified and designed in a platform-independent way, to support an abstract scenario of interaction between customers and suppliers, namely: (1) authentication; (2) business entities management (typical CRUD operations for customers, suppliers, and products); and (3) customer-to-supplier roles detailed product orders. This example has been continuously used across the entire ProjectIT initiative's history to provide a concrete proof-of-concept of the implemented tools functionality. Each frame presented below reveals an important requirements specification fragment, encompassing different aspects of the ProjectIT-RSL language.

From a user point of view, the conversion between the previous description of MyOrders2 into a ProjectIT-RSL specification can be supported by the ProjectIT-Studio/Requirements tool, which offers a powerful and interactive experience for requirements specification, in which users are assisted by error reporting mechanisms and a comprehensive set of views while writing. This is achieved by constantly validating the input against the specification model. The frame presented below, Code 1, depicts the typical structure of a ProjectIT-RSL requirements specification document, stressing the several types of sections and the hierarchical relations between them, as previously illustrated in Figure 3.

Code 1 – ProjectIT-RSL structure example

```
1 Section Introduction (...) End Section

2 System "SystemName"

        2.1 Section Comments (...) End Section

        2.2 Section Imports (...) End Section

        2.3 Section Business Entities (...) End Section

        2.4 Section Functional Requirements

                2.4.1 Section Actors Declaration (...) End Section

                2.4.2 Section Actors Definition (...) End Section
                2.4.3 Section Operations Declaration (...) End Section

                2.4.4 Section Operations Definition (...) End Section
        End Section

        2.5 Section Non-Functional Requirements
                2.5.1 Section Security (...) End Section
        End Section

End System
```

After defining the SECTION INTRODUCTION, the user must specify the main application unit region, called SYSTEM context (Code 2), followed by the system's name (in this case MyOrders2). At this level (as well as in other inner nested levels), users can specify as many SECTION COMMENTS as they want because they are neither parsed nor stored, since they only provide an auxiliary annotation mechanism for enriching the specifications description. Besides the SECTION COMMENTS, at SYSTEM scope level, the user can also specify a SECTION IMPORTS (only

one per SYSTEM region). This specific section type is used to declare import entries, which when parsed by the Fuzzy Matching Parser (FMP), makes it invoke the Structural Parser (SP) import mechanism to include its contents in the document's underlying conceptual model.

Code 2 – ProjectIT-RSL SYSTEM scope

```
2 System "MyOrders2"

        2.1 Section Comments
                The will describe the domain view, corresponding to the
                static/structural view of the system.
        End Section

        2.2 Section Imports
                - use \Documents\Document.
                - use \Documents\Invoice.
        End Section
(...)
```

The SECTION BUSINESS ENTITIES (Code 3) is of extreme importance because it is where one can define the domain model of the system currently under specification.

Code 3 – ProjectIT-RSL SECTION BUSINESS ENTITIES section

```
2.4 Section Business Entities
    <business entities pre edit point>

    2.4.1 A thirdParty is an entity.
    2.4.2 Each thirdParty has a 'company name' and a list of affiliates.

    2.4.3 An affiliate is an entity.
    2.4.4 Each affiliate has an 'affiliate type', and a 'contact name', and an
          address, and a 'contact title'.

    2.4.5 An address is composed by a city, and a 'postal code', and a
          country, and a phone, and a fax.

    2.4.6 An 'affiliate type' has a description.

    2.4.7 A supplier is a thirdParty.
    2.4.8 Each supplier has a list of 'order details', and an integer quality,
          and an integer responsiveness.

    2.4.9 A customer is a thirdParty.
    2.4.10 A customer is the same as a client.
    2.4.11 Each customer has an integer importance and an integer flexibility.

    2.4.12 Each customer has one or more markets.
    2.4.13 Each market has a list of markets.

    2.4.14 A market is an entity.
    2.4.15 A market has a name.

    2.4.16 Each supplier provides a list of products.
    2.4.17 Each product is provided by one or more suppliers.

    2.4.18 A product is an entity.
    2.4.19 A product has a 'product name', and a double 'unit price', and an
           integer 'order units', and an integer 'stock units', and boolean
           'discontinued'.

    2.4.20 A customer emits a list of orders.
    2.4.21 Each order is emitted by one customer.

    2.4.22 An order is an entity.
    2.4.23 An order has an 'order code', and an integer 'order number', and a
           date 'order date', and a date 'required date', and a date 'shipped
           date'.

    2.4.24 An order is composed by a list of "order details".

    2.4.25 An 'order detail' is an entity.
    2.4.26 An 'order detail' has a supplier, and an integer quantity, and a
           double 'unit price', and a double discount.

    <business entities post edit point>
End Section
```

Code 4, presents MyOrders2 user roles (or actors, in the ProjectIT-RSL terminology) hierarchy specification, as well as the features (currently, in the form of pre-defined actions, such as typical CRUD operations) involved on.

Code 4 – ProjectIT-RSL FUNCTIONAL REQUIREMENTS section

```
2.5 Section Functional Requirements

    // actors and operations declarations, operations definition

    2.5.1 Section Actors Declaration
        2.5.1.1 The User is an actor.
        2.5.1.2 The URegistered is a User.
        2.5.1.3 The UManager is an URegistered.
        2.5.1.4 The UAdministrator is an URegistered.
        2.5.1.5 The Person is the same as User.
    End Section

    2.5.2 Section Actors Definition
        2.5.2.1 UManager can create, and edit, and delete a customer.
        2.5.2.2 URegistered can create a customer.
        2.5.2.3 The User can create, and read, and update, and edit, and
                delete a thirdParty.
    End Section

End Section
```

When the Requirements Engineer finishes the specification of the requirements document, a corresponding UML 2.0 model can be produced automatically. Currently, only the domain model (class diagram) and actors hierarchy (a part of use case diagrams) can be generated. This model can be further refined during the design phase by using other ProjectIT-Studio tools, as depicted in Figure 5.

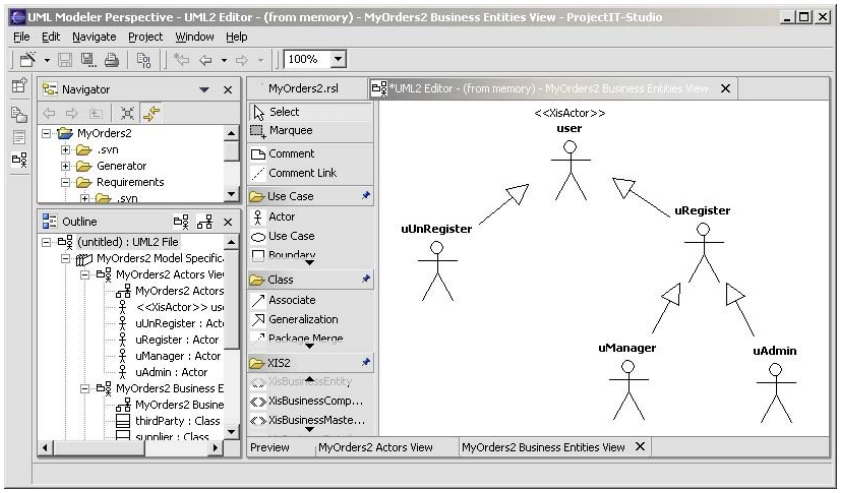

Figure 5 – Requirement UML model refinement

6. Conclusions

This paper presents our approach to deal with Requirements Engineering issues, improving the software development process as a whole when integrated with the remaining tools of the ProjectIT-Studio workbench. Our vision is realized into a requirements specification language, created upon the identification of common natural language linguist pattern of this kind of documents, and a supporting CASE tool, which offers several productivity enhancements. To better explain the ProjectIT-Requirements concepts and to demonstrate its respective support, the paper introduces the MyOrders2 academic proof-of-concept example, which shows in a simple but practical way how to define an information system's specification. We can observe that, despite it still lacks some functionality to improve users interaction, namely to provide more contextualized and user friendly information to enhance the validation process, this component already allows the specification of a considerable part of the target system's domain model and actors hierarchy. Thus, the current set of features offered by this tool is noteworthy when considering the inherent complexity of NLP. There are many other issues, questions, and research to be done in this context. The first key issue at the stage of our research is to enhance the language's extensibility and reuse mechanisms, namely providing procedural and workflow primitives. Second, the issue of components integration is also a mandatory one since we are seeking to attain an ontology or metamodel to integrate all the languages manipulated by the ProjectIT-Studio tools and, simultaneously, to enhance the artifacts traceability. Third, we have to continuously improve the NLP techniques, specifically the shallow parser and part-of-speech tagger, for enhancing the knowledge capture from the natural language specifications.

References

Abbot, R. (1983). Program design by informal English description. In *Communications of the ACM*, pages 882–894.

Adolph, S., Cockburn, A., and Bramble, P. (2002). *Patterns for Effective Use Cases*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Ambriola, V. and Gervasi, V. (2003). *The Circe approach to the systematic analysis of NL requirements*. Technical Report TR-03-05, Pisa, Italy.

Bell, T. and Thayer, T. (1976). Software requirements: Are they really a problem? In *ICSE '76: Proceedings of the 2nd international conference on Software engineering*, pages 61–68. IEEE Computer Society Press.

Cockburn, A. (2000). *Writing Effective Use Cases*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Cybulski, J. (2001). *Application of Software Reuse Methods to Requirements Elicitation from Informal Requirements Texts*. PhD thesis, School of Computer Science and Computer Engineering, La Trobe University.

Dutta, S., Lee, M., andWassenhove, L. V. (1999). Software engineering in europe: A study of best practices. *IEEE Software*, 16(3):82–90.

Fowler, M. (2007). Martin fowler - domain specific language. Retrieved Monday 3 September, 2007 from http://www.martinfowler.com/.

Fuchs, N. and Schwitter, R. (1996). Attempto Controlled English (ACE). In *1st Int. Workshop on Controlled Language Applications*, Leuven, Belgium.

Mich, L. and Garigliano, R. (1999). The NL-OOPS Project: OO Modeling using the NLPS LOLITA. In *4th Int. Conf. Applications of Natural Language to Information Systems*, pages 215–218.

Schmidt, D. (2006). Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31.

Silva, A. (2004). O Programa de Investigação ProjectIT (whitepaper). Retrieved Monday 5 June.

Silva, A., Videira, C., Saraiva, J., Ferreira, D., and Silva, R. (2006). The ProjectIT-Studio, an integrated environment for the development of information systems. In *Proc. of the 2nd Int. Conference of Innovative Views of .NET Technologies (IVNET'06)*, pages 85–103. Sociedade Brasileira de Computação and Microsoft.

Sommerville, I. and Kontonya, G. (1998). *Requirements Engineering: Processes and Techniques*. John Wiley & Sons, Inc., New York, NY, USA.

Standish Group International (1994). The Chaos Report. Retrieved Tuesday 4 September, 2007 from http://www.standishgroup.com/.

Videira, C., Ferreira, D., and Silva, A. (2006a). A Linguistic Patterns Approach for Requirements Specification. In *EUROMICRO '06: Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 302–309, Washington, DC, USA. IEEE Computer Society.

Videira, C., Ferreira, D., and Silva, A. (2006b). Using linguistic patterns for improving requirements specification. In *Proc. of the Int. Conference on Software and Data Technologies (ICSOFT 2006)*. INSTICC Press.

Videira, C. and Silva, A. (2004). ProjectIT-Requirements, a Formal and User-oriented Approach to Requirements Specification. In *Actas de las IV Jornadas Iberoamericanas en Ingeniera del Software e Ingeniera del Conocimiento*, pages 175–190 Volume I, Madrid, Spain.