Modelling of Enterprise Portals with Domain-Specific Language

Dominik Wasilewski
This thesis is submitted to the School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author:
Dominik Wasilewski
E-mail: dominik.wasilewski@gmail.com

External advisor
Maksymilian Kowalczyk
IT-Dev Sp. z o.o.
Brucknera 25-43
51-411 Wrocław, Poland
Phone: +48 71 723 49 04

University advisors:
Lech Tuzinkiewicz, PhD.
Institute of Informatics
Wrocław University of Technology

Ludwik Kuźnianz, PhD.
School of Engineering
Blekinge Institute of Technology, Sweden

School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden
ABSTRACT

Enterprise portals are comprehensive solutions that enable centralized access to information and employees. They also support the business processes taking place in companies. The diversity of functionality offered by enterprise portals is the source of the complexity of the manufacturing process of such applications. Domain-Specific Languages (DSL) are a novel approach to solving problems associated with the software development. By limiting the possibilities of expression to the concepts related to a specific area Domain-Specific Languages are more focused on solving specific problems.

The subject of this thesis is DSL SharePoint – Domain-Specific Language which supports the production of enterprise portals on Microsoft SharePoint platform. Language was developed with respect to the newest achievements in area of building DSLs.

By applying the language in the industry, it was possible to verify the hypothesis that its usage positively affects the quality of software products. To this end, the quality model was built, and products made with the support of language have been compared to those developed in the traditional manner.

Keywords: Domain-Specific Languages, Enterprise Portals, software product quality, software development tools.
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>32</td>
</tr>
<tr>
<td>4.2</td>
<td>SharePoint platform overview</td>
<td>32</td>
</tr>
<tr>
<td>4.3</td>
<td>SharePoint architecture</td>
<td>33</td>
</tr>
<tr>
<td>4.4</td>
<td>SharePoint development techniques</td>
<td>34</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Customization versus Development</td>
<td>35</td>
</tr>
<tr>
<td>4.4.2</td>
<td>SharePoint object model and Web Parts</td>
<td>35</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Features</td>
<td>36</td>
</tr>
<tr>
<td>4.4.4</td>
<td>Site definitions</td>
<td>37</td>
</tr>
<tr>
<td>4.5</td>
<td>SharePoint data model</td>
<td>38</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Columns</td>
<td>38</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Field types</td>
<td>39</td>
</tr>
<tr>
<td>4.5.3</td>
<td>Content types</td>
<td>40</td>
</tr>
<tr>
<td>4.5.4</td>
<td>Lists and document libraries</td>
<td>42</td>
</tr>
<tr>
<td>4.5.5</td>
<td>Views</td>
<td>44</td>
</tr>
<tr>
<td>4.6</td>
<td>Issues in development process</td>
<td>45</td>
</tr>
<tr>
<td>4.6.1</td>
<td>Defining columns</td>
<td>45</td>
</tr>
<tr>
<td>4.6.2</td>
<td>Defining Lookup fields</td>
<td>46</td>
</tr>
<tr>
<td>4.6.3</td>
<td>Content type IDs and inheritance</td>
<td>46</td>
</tr>
<tr>
<td>4.6.4</td>
<td>Referencing columns in content types</td>
<td>47</td>
</tr>
<tr>
<td>4.6.5</td>
<td>Adding columns to lists</td>
<td>48</td>
</tr>
<tr>
<td>4.6.6</td>
<td>Adding content types to lists</td>
<td>48</td>
</tr>
<tr>
<td>4.6.7</td>
<td>Defining views</td>
<td>49</td>
</tr>
<tr>
<td>4.6.8</td>
<td>Defining features</td>
<td>50</td>
</tr>
<tr>
<td>4.6.9</td>
<td>Defining sites</td>
<td>50</td>
</tr>
<tr>
<td>4.6.10</td>
<td>Working with SharePoint collections</td>
<td>52</td>
</tr>
<tr>
<td>4.7</td>
<td>Summary</td>
<td>53</td>
</tr>
<tr>
<td>5</td>
<td>SharePoint DSL</td>
<td>55</td>
</tr>
<tr>
<td>5.1</td>
<td>Introduction</td>
<td>55</td>
</tr>
<tr>
<td>5.2</td>
<td>Description of tool</td>
<td>55</td>
</tr>
<tr>
<td>5.3</td>
<td>Description of process</td>
<td>57</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Decision</td>
<td>57</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Analysis</td>
<td>57</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Design</td>
<td>58</td>
</tr>
<tr>
<td>5.3.4</td>
<td>Implementation</td>
<td>61</td>
</tr>
<tr>
<td>5.3.5</td>
<td>Deployment</td>
<td>65</td>
</tr>
<tr>
<td>5.4</td>
<td>Mechanisms for solving problems</td>
<td>67</td>
</tr>
</tbody>
</table>
Attractiveness Survey........................................................................................................... 115
Ease of Use Survey ........................................................................................................... 116
1 Introduction

1.1 Background

Companies of all sizes benefit from large variety of software which supports theirs activities. Starting with the basic applications dedicated for work in office (Microsoft Office, OpenOffice), across Content Management Systems (CMS) which is used to create, edit, manage and publish a variety of content in consistently organized fashion [6] or Document Management Systems (DMS) which are designed to assist in creation, storage, retrieval and expiry information stored as documents [6], throughout large systems which supports Enterprise Resource Planning (ERP), or Customer Relationship Management (CRM). The large variety of software used in enterprises brings forward the need for place, where distributed information may be integrated and easily accessed by all employees. An Enterprise Portal (EP) also known as Enterprise Information Portal (EIP) is a framework for integrating information, people and processes within organization [2]. The Enterprise Portal is often complex, Web application, which allows not only access to the variety of enterprise resources, but also contribution to it, so that the information can be managed in decentralized way. However, EP’s can be classified not only as corporate Content or Document Management System. Portal should also allow project collaboration, business process management (usually with workflows), information management (access to data warehouse and related business intelligence repositories) and any other kind of functionality, which may assist processes within the enterprise. Most Enterprise Portal vendors, such as Microsoft (Microsoft Office SharePoint Server 2007), IBM (WebSphere Portal 6), Oracle (WebCenter Suite) or SAP (NetWaver 7), deliver portals with basic functionality, which can be expanded and customized to fit specific needs of a customer. There are however cases, when customization is insufficient and development of additional components is needed. Although development brings more flexibility and adaptability to the portal, it has many disadvantages associated with development process, such as time consumption or high costs.

Domain-Specific Language (DSL) can be defined as "a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain." [36] DSLs are often considered as an integral part of Domain-Specific Development, which is based on the observation that many software problems can be more easily solved by designing a small, special-purpose languages [3]. Usage of Domain-Specific Languages can bring many advantages to software development process. According to [36] it enhances reliability, reusability, quality, and flexibility, provides self-documenting and portable descriptions, reduces forward (and backward) engineering costs and facilitates timely delivery of software systems. Models expressed in DSL may be validated at the level of abstraction of the problem space. Thanks to that, errors may be picked up early in development process. Models can be also used as simulation of solution, which allows project stakeholders to give early feedback concerning proposed solution [3].

The Domain-Specific Language developed for modelling certain aspects of Enterprise Portal can improve the process of development in many areas. Well-designed DSL may allow validating the model in early stages of the project. Graphical
notation may provide new means of communication between developers and users. Generation of code directly from the model may increase overall quality of a product and reduce its development costs. Therefore development of DSL for EP platform may by very profitable, considering the fact that Enterprise Portals are often complex systems.

1.2 **Aims, objectives, and research questions**

The main aim of this thesis was to investigate the change of quality of SharePoint software products. This aim was achieved by constructing Domain-Specific Language which purpose is modelling of Enterprise Portals on SharePoint platform. Usefulness of this language was evaluated basing on software products, which were developed with and without its support. In order to achieve it, the following objectives were fulfilled:

O1. Review current state of knowledge about DSLs and EPs.
O2. Investigation of development process for DSLs.
O4. Design and development of DSL for SharePoint platform.
O5. Comparison of software products developed with and without usage of DSL.

The following research questions were answered during the course of this work:

Q1. What has been proposed in literature about development of DSLs?
Q2. How the process of development of DSL is conducted?
Q3. How to improve the quality of EP on SharePoint platform?
Q4. What are the key aspects of EP that are suitable for creating DSL?
Q5. How the usage of DSL during development of EP affects the products’ quality?

1.3 **Methodology**

This work has background in two areas – Enterprise Portals and Domain-Specific Languages. Those areas were subject of literature study, which aim was to acquire and systemize knowledge associated with them. Literature which was reviewed includes acknowledged reports, articles and books associated with particular areas. The study focuses on definition and scope of Enterprise Portals, as well as definition and methods of development for Domain-Specific Languages. Literature study is method for fulfilling objectives O1 and O2, and provides answers for questions Q1 and Q2. Next step was to define problems associated with portal development. Those problems were gathered during author’s professional work, and along with presentation of development process for SharePoint portals, are fulfilment of objective O3 and answers questions Q3 and Q4. Next objective (O4) was to design and develop actual language, which main purpose is to solve defined problems. The outcome of this phase was an answer to the research question Q4. During the last phase of the project, language impact on the quality of portals was evaluated, by designing quality model for SharePoint software products and applying it to products developed with and without support from DSL. Quality was evaluated by using both qualitative and
quantitative methods, conducting test, performing survey and analysing data associated with SharePoint portal development.

1.4 Motivation

Motivations for this work were problems and issues encountered during professional work of author of this thesis. Solutions presented in this work are parts of framework, which supports software development in IT-Dev sp. z o.o. – software development company from Wroclaw. It specializes in development of Enterprise Portals on Windows SharePoint Services platform. Work with this platform is difficult and expensive, because of its complexity and wide range of functionality it offers. On the other hand, there is a strong need and wide market for portal solutions. Elimination of obstacles associated with development process, which are presented later in this work, had positive impact on developers’ productivity, and enabled fulfilment of more contracts in given time. It also increased quality of supplied products.

Another motivation for choosing the topic for this thesis was an opportunity to solve real life problem. Scientific works that are strongly connected to industry have a great value, mainly because of the fact, that they can be instantly applied to particular field. Close relation between real life problem and scientific work, in lots of cases guarantees that work will be applied and verified in practical scenarios. Moreover, this kind of work is more likely to be continued and extended. The support that can be provided by both scientific and industrial advisors was also important factor.

1.5 Structure

This work consists of seven chapters where first and last chapters are respectively introduction and summary. Chapter 2 and 3 are result of research of background of this work, which was based on acknowledged literature. Both chapters are detailed description of areas, which are referenced in this work. Chapter 2 presents area of the problem, which is Enterprise Portal development, whereas chapter 3 - area of solution, which is Domain-Specific Language. In both chapters, first sections provide definitions, on which problem definition and solution will be based. Portal description is supplemented by description of development techniques and problems associated with development process. Apart from Domain-Specific Languages definition, they are also described considering their advantages and disadvantages, various kinds of their classifications and their development processes.

Chapter 4 provides detailed definition of the problem, which is based on contents of chapter 2, experience of author of this work and additional study of literature, which is associated with SharePoint development and the platform itself. It provides brief introduction to the platform, its architecture and development techniques, which may be applied during constructing SharePoint-based products. It also delivers definitions of SharePoint-specific entities, which are related to portal development, along with their purpose, methods of representation and methods for their development. In next section, issues associated with development of those entities are described in detail, providing set problems, which are solved by proposed DSL.

Chapter 5 presents solution for problems defined in chapter 4, which is provided by Domain-Specific Language. It starts with presentation of means for language design
and implementation, by introducing tool which was used during the process. Next section provides detailed description of each phase of DSL development process, with respect to its description provided in chapter 5. It consists of deciding whether language is really necessary, its design, implementation and methods of its deployment. In next section, techniques which were used in order to solve particular problems are described in detail, by presenting appropriate part of definition, example of its usage and short description of solution.

Chapter 6 contains assessment of usefulness of the language, which consist of building and applying quality model for evaluation of SharePoint software products. Model definition was preceded by description of the method, which was used to develop it, and which was based on upcoming ISO 25010 standard. Various software products’ aspects, introduced in chapter 6, were evaluated basing on several sources of data, such as result of a survey, outcome of research and analysis of data associated with SharePoint development.

Chapter 7 summarize this work by providing its outcome and the value that was introduced by it. It also presents problems, which occur during project realization and its shortcomings. Additionally, it provides set of issues, which may be basis for future work.

Structure of this thesis, which includes chapters and relationships between them, is shown on Figure 1.
Figure 1. Structure of the thesis
2 Enterprise Portal

2.1 Introduction

Enterprise Portals are first area, which is described in detail. This area is source of problems, which are solved in this work, therefore its proper definition, along with declaration of its scope is crucial. This chapter consist of following sections:

- Section 2.2 provides definition for term Enterprise Portal, as well as terms associated with it, such as Portal. Those two terms are closely related, thus Enterprise Portal definition is based on definition provided for Portal. Additionally, it presents several different approaches to defining portal, from which one which is most suitable for this work was chosen.

- Section 2.3 describes various techniques and approaches to development of Enterprise Portal, along with their comparison. It also lists their main advantages and disadvantages. This background is necessary for understanding problems, which occurs during development.

- Section 2.4 presents overview of issues that occur during portal development process. This high-level overview will be specified in more details in chapter 4.

- Section 2.5 summarizes this chapter.

2.2 Definitions

2.2.1 Portal

“Portal” is the term, which is ambiguous and overloaded - its precise meaning depends on the context in which the term is used. In most cases, portal is a public place, where information is stored and shared. Examples of portals are web sites such as Yahoo, America Online, or MSN (see Figure 2). Those public portals had begun to evolve in mid-1990s. At the beginning, their functionality was limited to a set of key features such as news, e-mail account and search. Their main purpose was to grant access to those resources with consistent look and feel. From the beginning, their main function was to “provide as much information and services as possible in one place.” [2] Over the time, enhancements such as authentication and personalization were introduced into portals, making them more user-oriented – adapted to specific users’ needs. Today’s portals are starting point to the Internet for millions of users all over the world.
Figure 2. Example of public portal

More technical definition of Portal can be found in [11]: “A portal is a web based application that – commonly – provides personalization, authentication, and content aggregation from different sources and hosts the presentation layer of information systems.” Content aggregation is provided by integrating information from different sources, while personalization and authentication - by sophisticated features and the interaction with user, which is realized as a set of portlets. Portlet is defined in the same standard [11] as “an application that provides a specific piece of content (information or service) to be included as part of a portal page.” They are usually pluggable user interface components, which interacts with the rest of the system basing on request/response paradigm. Pages of portals usually consist of several portlets, which provides information from various sources. By storing portlets in one place, information, data or services may be accessed in centralized way.

2.2.2 Enterprise Portal

Companies of all sizes realized, that functionality provided by public portals can be relatively easily adapted to their needs. Motivation that stands behind the incorporation of portals into the business was need for the access point to the various company resources. As the portals evolved, so does the Enterprise Portals, by incorporating more functionality. Today’s Enterprise Portals are access point to data, services and other applications, which enable communicating and collaborating with other employees. Figure 3 presents home page Enterprise Portal developed by IT-Dev, which provides access to different kind of information.
Figure 3. Example of Enterprise Portal

The wide range of functionality, provided by EPs is one of the reasons of misuse of the term “Enterprise Portal”. Basically, there are three kinds of definitions of Enterprise Portal, which differs mainly in range of services provided by them [7]. The first class defines portals as “applications that enable companies to unlock internally and externally stored information, and provide users a single gateway to personalized information needed to make informed business decisions.” [32] Similar approach is presented in [38], where EP is defined as “a single Web interface to corporate information scattered throughout the enterprise.” Both definitions put strong emphasis on the fact, that portals should provide access to various kinds of information (internal and external, structured and unstructured). On the other hand, alternative definitions of EP focus more on its users, collaboration between them and workflows – they do not emphasise information processing. In [30] accent is placed on “integrating the islands of automation formed by today’s application-based desktops.” It means that Enterprise Portals should create environment, which provide information access and delivery along with work support. The third interpretation of Enterprise Portals defines them more generally, by combining information and collaboration processing perspectives. This approach was presented in [26], where author stated that portals that focus only on content will not meet company’s needs. Instead, EP should “connect us not only with everything we need, but with everyone we need, and provide all the tools we
need to work together.” Along EP definition, [29] provides seven defining characteristics, which distinguish EP:

- Integrating access in a wider variety of data formats than a Web portal (comprehensive);
- Organizing access to information for users to browse (organized);
- Assembling personalized views of key information and notifying users of the availability of new material via electronic mail and other media (personalized);
- Organizing access to data, but not storing the data itself (location-transparent);
- Supporting extensions for cataloguing new types of information (extensible);
- Automatically identifying and organizing access to new content (automated);
- Selectively brokering access to internal corporate information (secure).

Those various definitions of the term “Enterprise Portal” have been cited in order to show the different understanding of EPs. In most cases, authors of narrow definitions, which focus on only one group of the functionalities, that may be the part of EP (information management in [32] and [38] versus collaboration in [30]) do not deny the fact, that portals may be expanded. Usually, portals that have more functionality or are focus on solving other class of problems, have different names (Corporate Portal, Collaborative Portal, Enterprise Knowledge Portal etc.). This excess of definitions, called “definition cacophony”, is the main source of confusion while talking about EPs.

In this work, the Enterprise Portal will be defined with respect to the third understanding of the term. The Enterprise Portal is both a gateway to the various repositories of information within the company, and a platform which supports collaboration and process management. Platform will be defined as whole set of hardware and software components, which allow EP software to be run. The best definition, which incorporates those two concepts, was presented in [2], where Enterprise Portal was defined as “A framework for integrating information, applications, and processes across organizational boundaries.”

2.3 Development techniques

Since 1999, the software vendors began releasing so called “portal software” – toolkits for developing portals [2]. Nowadays, every portal vendor offers a complex application with both wide range of built-in functionality, along with possibility of expanding it, in order to fit to specific customer needs. There are many techniques of expanding and tuning portals, and each of them has their own advantages and disadvantages. The main difference, which distinguishes those methods, is the degree of programmers’ involvement in portal development process, which is related to level of abstraction. The higher level of abstraction, the less programmers work is needed. On the other hand, working on the lower level of abstraction may introduce more flexibility and adaptability into both development process and final product.
Figure 4. The Portal Development Curve, based on [2]

Chart on Figure 4 presents characteristics of portal development techniques as a relationship between level of abstraction and flexibility. Some of the techniques (wizard-based development, solution packages and customization) require little or no programming work. Portal may be tuned by administrators, who install portal software from various kinds of packages or templates, and choose desired options. Portals may be also customized by their user, who chooses the components that appears on their personalized web pages. Those techniques are easy to use and may be quickly applied, but they are not flexible enough for fulfilling more sophisticated requirements. That is the reason for existence of the second group of techniques, which requires providing custom portal components (portlet development) or writing code against frameworks, APIs and Web Services.

2.4 Development issues

Enterprise portals are complex type of software which offers wide range of functionality. On the other hand, complexity is main source of difficulties in portal development. Issues may appear in different project areas and during different stages of development process. Nevertheless, elimination or even reduction of some problems would greatly reduce an effort necessary to deliver EP product.

Like in many other software products, not only those related to portals, the main issue associated with development is the fact, that many programming tasks are similar or follow the same pattern. There are also other recurring activities, such as assembling components from common set of assets. In order to exploit those similarities, mechanisms for automatic source code generation may be introduced into development process. This may also decrease complexity of the development process, which is caused by vast set of functionality, which should be part of the portal. In order to satisfy customers’ needs, the portal should be able to collaborate with other office applications, enable access to various data repositories, automate business processes etc. It means that portal should support integration with other applications, such as data management and workflows.
The second class of problems is associated with Enterprise Portal design. Although all of them are, by definition, Web-based applications, and usually follows Model-view-controller (MVC) architectural pattern [9], their design is not limited to determine relationships between presentation, business logic and data access layers. Those aspects are usually already determined by portal software vendors, so portal architects have to fit to existing architecture, by merging their design with the one provided by vendors. The design of Enterprise Portal can be seen as a definition of components, which provide to users various kinds of functionality, definition of structures in which data for the portal will be stored, project and design of Web sites that will be part of the portal and so on. Methods for defining those aspects depends on the portal software – each portal software vendor, provides its own set of APIs, frameworks, tools and instructions for development of specific portals. Architects may leverage those tools and techniques in defining design aspects of portal, but before that, they have to learn those “portal-specific” techniques and identify their advantages and weaknesses.

2.5 Summary

At the beginning, Enterprise Portals served as a source of entertainment, daily news or other information. In time, those functionalities were adapted and widened for the needs of business. Although EPs still have many commonalities with Portals, they offer much wider set of functionality than their predecessors. Today’s EP is not only intranet home pages for companies – it is a gateway to company’s resources, place for integrating employees, and tool which supports business processes automation. The importance of EP in companies make their development profitable, yet complex task. Enterprise Portals vendors ship their products in form of platforms, which enables creating products by working on different levels – from simple customization to development of custom components and working with underlying framework. Flexibility offered by development on lower level of abstraction is associated with longer and more difficult development process. Therefore, any kind of improvement is desired. Those improvements should solve problems such as eliminating repetitive tasks, and assist during both portal design and development phases.
3 Domain-Specific Language

3.1 Introduction

In this chapter, the second area of this work, which is Domain-Specific Languages, is described. This chapter is important for this work, because it presents principles of technique, which will be used in order to solve problems associated with Enterprise Portal development. Sections contained in this chapter provides background necessary for understanding the idea that stands behind DSLs, including information about what exactly Domain-Specific Languages are, what their role in development process is, when they may be used and how they may look like.

- Section 3.2, like in the previous chapter, presents various definitions of Domain-Specific Languages, and indicates one which will be applied in this work. Additionally, definitions of terms related to DSL, such as domain, domain expert or general purpose language, are presented as well.

- Section 3.3 presents various categorizations of Domain-Specific Languages. Those categories are associated with various aspects of DSL, such as form of the language, or its relation with existing language or notation.

- Section 3.4 presents set of advantages and disadvantages of DSLs, which have to be carefully considered when deciding whether DSL should be developed or not. It also provides information about areas of software development, which may be improved by introducing DSL.

- Section 3.5 provides description of Domain-Specific language development process. There are no standardized methods or techniques of DSL development, so recommendations from several sources are gathered and revived in this section. Development process consist of phases and patterns associated with this phases, which are also described in this section. Development of DSL, which was one of the aims of this work, was conducted according to definition of the process contained in this section.

- Section 3.6 summarizes this chapter.

3.2 Definitions

3.2.1 Domain

Software is usually made for the specific customer or for the group of customers from within the same area of business. It is usually impractical, or even impossible to develop software, which could be applied in multiple areas. For example, there is no sense in developing software which controls Space Shuttles, monitors patients’ health and allows creating and editing documents. Each of the business areas has its own specific problems and needs, which have to be properly recognized in order to produce valuable software. The area of business, engineering or society where body of knowledge exists is called domain [4]. According to [25] the general definition of domain is “A sphere of activity, concern, or function; a field, e.g. the domain of history.” However the domain can be viewed not only as a field from the real world, but also as a set of systems. In [4] such kind of domain was defined as: “area of knowledge: scoped to maximize the satisfaction of the requirements of its
stakeholders; including a set of concepts and terminology understood by practitioners in that area; including the knowledge of how to build software systems in that area.” The examples of domains include administration, banking, accounting, healthcare, computer gaming etc.

3.2.2 Terms related to domain

There are several other terms, which are often used when speaking about domains. First is domain expert, who is “a person who has specific domain knowledge about the problem that is to be solved by software.” [12] The main activity for domain expert, during software development process, is communicating enough domain knowledge to other stakeholders, so that the software can be built. This communication is conducted with help of domain terminology, which can be defined as “terms of art (words with a special meaning) in a domain.” [12] It is very important for all project stakeholders to understand the domain terminology, because it is often only way to express intentions, and any misuse of words or misunderstanding could lead to the failure of the whole project.

3.2.3 General Purpose Language

The term General Purpose Language (GPL) is often used as an antonym to Domain-Specific Language. It describes all programming or modelling languages, which are not restricted to solving one particular kind of problems. For this reason, they can be applied to multiple domains in many different ways. This generality may be achieved in two ways – the first is to create complex language, which will provide means to describe many different systems and their aspects. The second approach, which is taken in most cases, is to create language, which basic components and constructs are very abstract, and they can be instantiated for particular cases. The suitable example of such construct may be class, which can be defined as collection of things or concepts, which have the same characteristics: static – attributes, and dynamic – operations [31]. Class in object-oriented programming language is a mechanism, which helps to describe any concept from real world. GPLs may be divided, considering their form, into two groups – general purpose programming languages, such as C, C++, C# or Java, and general purpose modelling languages such as UML. Most important advantage of GPLs is their capability of describing and solving almost every problem. Their drawback is complexity of programs or models created with them, and their complexity.

3.2.4 Domain-Specific Language

There are many definitions of Domain-Specific Language (DSL) which differs in details, but almost all of them describe DSL as a kind of language designed to express concepts from within specific domain. However, as mentioned before, there are many extensions to that basic definition. According to [36] and [3] DSL may be a programming language or a specification language or a design language. At the beginning, many considered DSL to be programming language only, but the still increasing popularity of graphical modelling languages and software design approaches such as Model-Driven Architecture (MDA) changed the overall approach
to DSLs and widened its definition. In [36] and [34] DSLs are described as more declarative than imperative languages. The [37] defines it as: “a high-level software implementation language”, so it should abstract both from low-level implementation details and platform. Following definition of DSL will be used in this work: a Domain-Specific Language is a programming, specification or modelling language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain. As not entirely new concept, Domain-Specific Languages in literature are also known as: little languages [35], micro-languages [36] or special-purpose programming languages.

3.3 Classification

DSLs can be divided and categorized in many ways. Language itself may take a form of graphical representation or textual notation or mixture of both [3] [34]. DSLs may be more or less specific, depending on the characteristics of domain that they were design for and the level of abstraction on which the domain was described. For example, DSL for expressing thread synchronization is more general than DSL for defining financial products [4], because it may be used in banking and other areas as well. It is also possible to categorize DSLs by their function: some of them are focused on communication with domain experts (Business Process Modelling Notation - BPMN); some of them are tools which allow creating and editing complex, XML-based configuration files, which are very common in modern software. There are also DSLs, which are designed for programming task, such SQL for databases or UNIX shell languages for communication with Operating Systems (OS).

3.3.1 Internal and External DSLs

There are two approaches to Domain-Specific Languages development. First strategy is to use existing languages to emulate DSL. The capabilities of modern programming languages, such as defining different kinds of abstractions, such as classes, structures, enumerations [3], makes it quite easy. The existing General Purpose Language, which is a base for DSL is often called host language. Tradition of building DSLs on the host language is quite old, considering so called micro languages developed within Lisp programming language [8]. This kind of DSL is called Internal DSL. While the Internal DSL is a part of host language, it is possible to benefit from full power of the language itself, as well as from the supporting tools. On the other hand, Internal DSL has the same limitations as the host language on which it is based. Additionally tools do not distinguish host language and DSL, so they do not support DSL in such tasks as validation against domain rules. Internal DSL are also difficult to learn by people who don’t know the host language. That is why Internal DSLs are more useful for the purpose of increasing efficiency of development process rather than communicating with domain experts. The issues with Internal DSL are summarized in [8]: “(…) a general purpose programming language gives you lots of tools - but your DSL uses only a few of these tools. Having more tools than you need often makes things harder - because you have to learn what all these tools are before you can figure out the few you use. Ideally you want only the actual tools you need for your job - certainly no less, but only a few more.”
The second approach to development of Domain-Specific Languages is to create them “from scratch”. Those languages, are completely independent from existing languages, and are called External DSLs. Their main disadvantage is necessity of building some kind of compiler or interpreter in order to transform DSL into language, which is recognizable by runtime environments, which is very costly and complex task. However popularity of DSLs caused emerging of tools, which supports their development by providing generic compilers or interpreters. The main advantage is the fact, that the language can take any form. It is useful especially in those cases, when some notations for domain already exist. External DSLs has “lot of ability to express the domain in the easiest form possible to read and modify” [8]. The only limitation associated with notation for the language is ability to create interpreter.

Both External and Internal Domain-Specific Languages have their advantages and disadvantages, and thus they may be considered complementary. For the purpose of collaboration between domain experts and programmers, External DSLs may be considered more suitable, however deep integration Internal DSLs with host languages makes them more useful for programmers.

### 3.3.2 Graphical and Textual DSLs

Domain-Specific Languages can be also divided by form of their notation, into graphical and textual. However, considering expression capabilities of language, there is no fundamental difference between those two types of notations. According to [37] “The essence of a language is that it defines structures to which meaning is assigned.” Both creating and reading those structures may be achieved with a variety of tools. Additionally, there may be many various representations of those structures, which are interchangeable. There are some differences in usability, because some notations are more suitable for some applications than the others. Every graphical diagram has its textual representation - for example rendering instructions contained in XML format. In the same way textual models may be “transformed” into graphical representation. It can be done by displaying the tree or graph structure, which is result of expression parsing followed by static semantic analysis [37].

Nowadays, most of the languages tend to be graphical rather than textual [37]. This tendency is caused by the fact that graphical languages are usually easier to read and understand than their textual counterparts. Graphical languages are more suitable for communicating with non-developers, especially with domain experts. The second reason is the fact that modern languages tends to describe problems and solutions on higher level of abstraction. This allows focusing on the business goals, rather than on technical details of solution. When there is no need for detail, textual descriptions, graphical representations are more suitable for showing so called “bigger picture” – a problems along with context and relationships between them.

Although textual Domain-Specific Languages might be difficult to interpret, most successful DSLs created up to date are textual [37]. The main reason is that they are more flexible than graphical languages - some complex dependencies between shapes on the diagram may be represented by just one line of code. Secondly, for experienced developers, it is easier and quicker to write some piece of code rather than drag and drop shapes and specify their properties. Additionally, development of graphical language of any kind is more difficult than its textual counterpart, because of lack of tools and known techniques for doing so. However, according to DSL definition
(paragraph 3.2.4), they may be modelling languages as well. The UML (Unified Modelling Language) diagrams, which are used to model aspects of software systems in Model Driven Architecture (MDA), may be tuned for particular domain by using UML profiles [37].

### 3.4 Advantages and disadvantages

#### 3.4.1 Advantages

The usage of Domain-Specific Languages offers many opportunities in the field of improving process of software development and quality of the software itself. One of the prominent advantages of DSLs is the fact that it allows communication between domain experts and programmers in terms taken directly from domain. Thanks to that, people with domain knowledge may take an active role in development process. Moreover “a DSL allows a computationally naive user to describe problems using natural terms and concepts of a domain with informality, imprecision, and omission of details.” [1] This may help avoid errors and misunderstandings, because the models are more readable for domain experts, and thanks to that any problems can be identified and solved in early stages of development process. Model can be also used as the simulation of the product, which enables immediate feedback on solution [3]. DSL can be viewed as the embodiment of domain knowledge, which is very handful for conservation and reuse [3]. Reusability can be also understood in context of breaking large general purpose languages into modular languages extensions, which can be used in different configurations [4].

From the developers’ point of view, DSLs provides domain specific API which increase their productivity. It is possible to generate different kinds of artefacts from DSL models – not only source code, but also configuration files, documentation, build scripts, bills of materials and many more [3]. Programming with usage of DSLs is also less error-prone in contrast to general purpose languages (GPL). In addition, it is easier to apply automated proof techniques when the language is small and specific [34]. It is also worth mentioning, that DSLs show in more understandable way a purpose of creating each part of a system, which results in better solutions. The other benefits of DSL are enhancement of maintainability [35], portability [36], scalability and fast feature turnover [4]. Using DSLs increases reliability and reparability, by providing “self-documenting and portable descriptions, and reduces forward (and backward) engineering costs.” [35]

#### 3.4.2 Disadvantages

On the other hand, the concept of Domain-Specific Languages is not without any weaknesses. The main is the cost of designing, developing and maintaining both language itself and supporting tools [36]. At first, decision in favour for DSL development must be carefully considered, because in some cases domains are specific and narrow. In such cases, language would not be used in more than few projects, and the cost may outweigh potential benefits. It may be also difficult to find proper scope for DSL, and in the end it will be too narrow or too complex to be useful. Some problems are associated with the usage of DSLs. In some cases, created DSL
may be too complex for the domain expert, which disqualifies one of its main advantages. In other cases, usage of DSL can make a system “more difficult to maintain, for example if changes to the underlying domain model become necessary.”

[35]

In general, disadvantages of Domain-Specific Language are costs of its development, including gathering domain knowledge, design, implementation, and building the tools necessary to use it [8]. They are however major step towards innovative software development model, in which domain expert is the person who is responsible for business logic of the system. The programmer task would be tuning and customizing the systems, but most of all, creating support tools such as DSLs and application generators.

3.5 Development process

The popularization of Domain-Specific Languages brings the need of standardization the DSL development techniques. Until recently, no systematic approaches to DSL development were applied, or they were developed along with DSL itself. However, there are several reviews performed in papers such as [33] and [17]. In those works, patterns associated with development of DSLs are identified. Other works, such as [37] or [16], presents overall approaches to DSL development, or identifies requirements which useful DSL must fulfil. According to [17], DSL development process consists of five phases: decision, analysis, design, implementation and deployment. There are a number of patterns associated with each development phase. Those patterns are independent, so there are many possible variants of DSL development process. Additionally, patterns associated with the same development phase, usually have some overlap. There are others patterns classifications, such as the one presented in [33]. The author closely follows the patterns classification presented in [9], and divides DSL patterns into three classes: creational, structural and behavioural. Those classes share some commonalities with development phases. The main difference between those classifications is that Spinellis consider the process and economics, that lay behind realization of a DSL, completely different that those of traditional programming languages. This viewpoint is in contradiction with presented in [17], where author consider General Purpose Languages design and implementation techniques relevant for Domain-Specific Languages as well.

3.5.1 Decision

As it is stated in [37] "The development of a DSL starts with the decision to develop one in the first place." It is often not easy to decide in favour of creating new DSL - the cost of developing it cannot outweigh potential benefits, so all pros and cons must be taken into consideration. Other possibilities, such as libraries, frameworks or external tool should be considered as well. In general, the inspection of the regular software engineering processes should take place at the beginning of the project. If those techniques fail to provide a right conception, especially abstract conception, developing DSL may be taken into account [37]. Additionally, there are kinds of problems, where creating DSL is recommended, because there are some
documented cases, where introducing DSL into development process solved those problems. Those patterns for problems are listed in Table 1.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notation</td>
<td>Add new or existing domain notation.</td>
</tr>
<tr>
<td>AVOPT</td>
<td>Domain-Specific Analysis, Verification, Optimization, Parallelization, and Transformation.</td>
</tr>
<tr>
<td>Task automation</td>
<td>Eliminate repetitive tasks.</td>
</tr>
<tr>
<td>Product line</td>
<td>Specify member of software product line.</td>
</tr>
<tr>
<td>Data structure representation</td>
<td>Facilitate data description.</td>
</tr>
<tr>
<td>Data structure traversal</td>
<td>Facilitate complicated traversals.</td>
</tr>
<tr>
<td>System front-end</td>
<td>Facilitate system configuration.</td>
</tr>
<tr>
<td>Interaction</td>
<td>Make interaction programmable.</td>
</tr>
<tr>
<td>GUI construction</td>
<td>Facilitate GUI construction.</td>
</tr>
</tbody>
</table>

Table 1. Decision patterns, based on [17]

Notation pattern is used, when the Domain-Specific notations already exist or there is a strong need for creating them. It may be further divided into two sub patterns – first of them is transformation visual notations into textual which may be useful in making large programs or documentations. The second sub pattern is simply adding more user-friendly form to the existing API or turning API into DSL.

Domain-Specific Analysis, Verification, Optimization, Parallelization, and Transformation (AVOPT) are often complex tasks, because GPL code patterns are (in most cases) not suitable for them. When aspects such as parallelization or performance are crucial, creating DSL may be a good solution.

Another pattern associated with code is task automation, which is very useful in increasing programmers’ productivity. The programming tasks, which follow the same pattern, have some common aspects or parts, may be replaced by automatic code generation. Thanks to that, time of the developers may be spent on more creative, business problems oriented tasks. Most basic examples of languages that follow that pattern are code snippets.

When assembling a Software Product Line (SPL), its members are built from a common set of building blocks, called assets. Those assets differentiate a lot, depending on the characteristic of the particular SPL. Nevertheless, creation of the product from SPL consists of choosing appropriate assets and defining relationships between them, which can be achieved by using DSL dedicated for assembling products from SPL. Additionally there are several notations for defining SPLs, which are also sorts of DSL.

Data structure representation and traversal are patterns useful in designing and maintaining complex data structures and traversals over them. Some frameworks define their own model for storing data, which are similar to the traditional ones only to some extent. DSL may prove to be useful assets for defining data structures accordingly to those unusual models.

System front-end pattern is useful in configuring complicated software systems, which often offers hundreds of configuring options. As stated in [17] the excess of features and configuration options can hide the real functionality. To avoid that, the DSL, which allows declarative and organized mechanisms for configuration, may be developed.
The last two patterns – interaction and GUI construction, simplify the design and implementation various kinds of interaction between user and software. Notable example of DSL which follows that pattern is Excel macro language.

3.5.2 Analysis

During analysis phase the domain knowledge is gathered and the problems from within the domain are identified. The input for analysis may be in form of implicit or explicit domain knowledge provided by technical documentation, domain experts’ knowledge, existing software or surveys. To achieve that, close collaboration between language developers and domain experts is required. The output of the analysis consists of domain-specific terminology along with its semantics. It is worth mentioning, that domain analysis is related to the knowledge engineering, so existing tools and framework may be adapted in order to support this phase.

The analysis phase is, by many means, crucial for the whole process. Every error, which occurs in this phase, may have disastrous consequences, which surface during usage of developed language. The next issue is that DSL should be as simple as possible, so language developers have to choose only fundamental tasks and establish appropriate abstraction of domain. In most cases there is no possibility of solving all problems identified within the domain. Table 2 presents patterns associated with this phase.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Informal</td>
<td>The domain is analysed in informal way.</td>
</tr>
<tr>
<td>Formal</td>
<td>A domain analysis methodology is used.</td>
</tr>
<tr>
<td>Extraction from code</td>
<td>Mining the domain knowledge from legacy GPL code by inspecting it or by using tools.</td>
</tr>
</tbody>
</table>

Table 2. Analysis patterns, based on [17]

It is possible to acquire domain knowledge ‘ad-hoc’, without using any methodology (Informal pattern), yet when there are some methodologies which suites the particular case, they should be used. The output of formal domain analysis is domain model often consisting of: definition (scope), terminology (vocabulary), descriptions of domain concepts and feature models describing commonalities, variabilities and interdependencies of domain concepts. If the legacy system exists, the extraction of domain knowledge from it is worth considering, especially when their users consider them to be sufficient in area of business functionality, and the need for developing new software is caused by new, non-functional requirements.

3.5.3 Design

During the design phase, the shape of the language and approach to its development process are determined. The design of DSL can be viewed from two different perspectives. The first perspective describes the relationship between DSL and existing languages, the second – formal nature of the design process. Patterns associated with those perspectives are presented on Table 3.
<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language exploitation (Internal DSL)</td>
<td>DSL uses part of existing GPL or DSL. Important sub patterns: Piggyback (language is partially used) Specialization (language is restricted) Extension (language is extended)</td>
</tr>
<tr>
<td>Language invention (External DSL)</td>
<td>DSL is designed from scratch with no connections to other languages.</td>
</tr>
<tr>
<td>Informal</td>
<td>DSL is designed informally.</td>
</tr>
<tr>
<td>Formal</td>
<td>DSL is designed formally using existing semantics definition methods such as attribute grammars, rewrite rules, or abstract state machines.</td>
</tr>
</tbody>
</table>

Table 3. Design patterns, based on [17]

As it was mentioned in section 3.3.1, the DSL may be based on the existing language (Internal DSL), or may be created from scratch (External DSL). In case of designing the DSL with relationships with existing languages, there are three methods of utilizing host languages. Piggyback is to introduce some domain-specific features into host language, while Specialization restricts existing language in order to hide those features, which makes it ambiguous (taking the domain needs into account). Those features often include aspects which are considered to be “unsafe” for inexperienced programmers such as dynamic memory allocation, unbounded pointers, or threads [33]. The third pattern is Extension - extending existing language with new features that addresses domain concepts. It can be also described as extending existing languages’ semantic and syntactic framework to the form of DSL [33]. As stated in [17] the main issue of this approach is choosing way of integrating new, domain-specific features with the rest of the language.

In [17] the development of External DSLs is considered to be extremely difficult to characterize. There are however some examples of possible approaches to External DSL development, such as described in [33]. DSL can be designed with the existing language development tools, and then by using source-to-source transformation DSL code can be transformed into GPL code or even into executable form. It is also possible to design DSL as a modelling language, and use model-to-model transformation, with UML model as an outcome of this transformation. Choosing external tool for designing DSL has many advantages, especially when it is dedicated particularly for DSL development. On the other hand, it often requires following patterns defined by the tool creators, thus limiting DSL architect options in some areas. Nonetheless, those constraints are usually outweighed by benefits of using existing tools, rather than building them from scratch.

The second set of design patterns for DSL includes informal and formal patterns. In an informal design there is no need for formal specification of DSL – the description in natural language along with example DSL programs is sufficient. Informal design is suitable for those DSLs, which are simple and developed for narrow domain. On the other hand, the formal specification written using one of the available semantic definition methods has its advantages. The most important is the fact, that it may identify the potential problems with DSL before it is actually implemented.
3.5.4 Implementation

Implementation is the phase, during which the actual DSL is created. Before that, the suitable implementation technique should be chosen. There are many implementation patterns which can be applied for textual or Internal DSLs (see Table 4). Those patterns are often based on existing programming language implementation techniques. There are also some implementation techniques, which are dedicated for implementing DSLs, and may not be useful for GPLs. Nonetheless there are little guidelines for implementing DSLs, which are intended to be independent from existing languages. Lack of methods and patterns to follow, applies also to situations, when graphical notation is chosen for DSL, and its main purpose is modelling, not implementation.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpreter</td>
<td>DSL constructs are recognized and interpreted using a standard fetch-decode-execute cycle.</td>
</tr>
<tr>
<td>Compiler/application generator</td>
<td>DSL constructs are translated to host language constructs and library calls. A complete static analysis can be done.</td>
</tr>
<tr>
<td>Pre-processor</td>
<td>DSL constructs are translated to host language constructs. Static analysis is limited to that done by host language processor.</td>
</tr>
<tr>
<td>Embedding</td>
<td>DSL constructs are embedded in host language by defining new abstract data types and operators.</td>
</tr>
<tr>
<td>Extensible compiler/interpreter</td>
<td>A host language compiler/interpreter is extended with domain-specific optimization rules and code generation.</td>
</tr>
<tr>
<td>Commercial Off-The-Shelf (COTS)</td>
<td>Existing tools and notations are applied to a specific domain.</td>
</tr>
<tr>
<td>Hybrid</td>
<td>A combination of the above approaches.</td>
</tr>
</tbody>
</table>

Table 4. Implementation patterns, based on [17]

According to [17] “interpretation and compilation are as relevant for DSLs as they are for GPLs”, but the costs and complexity of creating compilers or interpreters often discourage from using those techniques. Some languages, such as C++, support pre-processing approach in form of template metaprogramming. The template expansion can be used to achieve domain-specific code generation during compilation. Pre-processing approach can be also implemented as an expansion of macro definitions or translating the DSL code into host language code. The usage of listed patterns has many advantages – DSL syntax can be very close to the existing notations from domain, domain-specific analysis, verification, optimization, parallelization, and transformation (AVOPT) is possible, and the error reporting may be implementing on the level of the language. There are however some disadvantages such as necessity of building language processor, or difficulty in extending languages, which are usually not meant to be extended.

The embedding can be achieved by defining domain-specific abstract types and operators within host languages. The application libraries are basic form of embedding. The most suitable for embedding are functional languages, thanks to such
features as lazy evaluation, strong typing, polymorphism, or higher-order functions. Like in previous cases the embedding pattern has both advantages and disadvantages. Probably most useful feature which comes with this pattern is host language infrastructure (usually in form IDE) which can be reused in context of DSL. Besides, the training costs of DSL developers can be reduced, because they already know the host language, and some of the host language code can be reused. The major disadvantages of embedding are associated with confusion that may appear when using overloaded operators or during error reporting, because messages are in terms of host language rather than DSL.

Extending the host language compiler or interpreter with domain-specific optimization rules and code generation is usually complex task, particularly when compiler or interpreter was not designed with extensibility in mind. In many cases, the usage of existing tools and notations is suitable, which is the basis of Commercial-of-the-Shelf pattern. Restriction and customization of their functionality is performed in order to satisfy domain rules and needs. The most valuable, and most commonly used approach for implementation of DSLs, involves mixing features from other approaches, so the advantages of each of them can be exploited.

### 3.5.5 Deployment

Deployment is the phase of software development process, during which it is made available for use. It can be also described as a transition between states in which software creators were main stakeholder to the situation, in which software end-users are in that role. The result of successful deployment should be usage of the software, which solves users’ problems defined during requirement analysis. The deployment phase may include several activities, such as software release, installation, adaptation or actualization. Depending on needs of the consumer of the software, as well as its architecture, deployment may include different sets of activities. Unfortunately, deployment patterns or detailed description of the deployment process for Domain-Specific Language were described neither in [17] nor in [33], which are main sources for description of DSL development process. For this reason, patterns for DSL deployment will be proposed in this work (see Table 5). The character of DSL deployment is determined mainly by the tool, which allows programming or modelling using created notation. Without appropriate tool, it is impossible to work with DSL, thus the deployment phase could not take place. The character of the tool depends on the type of developed language.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension</td>
<td>DSL is deployed in form of extension to other tool or framework.</td>
</tr>
<tr>
<td>Tool</td>
<td>A stand-alone tool dedicated for creating DSL programs or models is created.</td>
</tr>
</tbody>
</table>

Table 5. Deployment patterns

The situation, when DSL can deployed as an extension for existing tool is probably most desirable for DSL developers, because in such situation they may take advantage of tool’s basic capabilities. Tools which are targets for extension are in most cases IDEs such as Visual Studio or Eclipse, because they already provide much functionality, such as rich text editors or visual designers, which may be leveraged for
DSL development. Additionally, most popular IDEs are built with support for extension, in order to provide third-party software vendors ability to expand those tools. This makes extension development much easier than building the tool from scratch. Finally, integrating DSL with tools familiar to people, who are target users of DSL makes transition and adaptation to different software development process quicker and easier. Software extensions may be distributed as plug-ins or add-ons (depending on target tool), thus they require presence of the host application. Extension must be built accordingly to requirements forced by host application or its extensibility framework. Additional advantage of DSL extensions is the fact, that some of the vendors provide ready-to-use tools for building IDEs extensions, which are dedicated for building DSLs. Examples of such tools may be Microsoft Visual Studio DSL Tools or Eclipse Modelling Project.

DSL follows Tool deployment pattern, when a stand-alone application is developed for DSL. This application should enable at least programming or modelling with developed DSL, which is most basic functionality for software development tool. However, it may provide much wider functionality which depends on the main purpose of the language. For example, when DSL was developed in order to improve communication between software producers and consumers, in most cases graphical notation is chosen for it. The tool, besides modelling capabilities, should enable saving diagrams in various graphical formats, validating diagrams against domain rules. It may be also shipped with interactive help or assistance, in order to improve non-IT users experience with this tool. The advantage of tools over the extensions is greater flexibility in choosing its functionality, which made those applications more DSL-oriented than IDEs with extensions. There is no need to follow any rules and restrictions that originate within host application, so tool may have any architecture, and its distribution among DSL users may take any form – from standard installation on local machine, to web application distributed in Software-as-a-Service (SaaS) model. On the other hand, building DSL editor, compiler etc. from scratch is more expensive and takes much more time than building it on the top of existing framework. Introducing new tool into software development process may also take more time, because developers have to learn to use it. Nonetheless, there are cases, when Extension pattern could not be applied, and development of tool is necessary.

Choosing the appropriate pattern for DSL deployment depends on the type of the language and its relation with existing languages. For Internal DSLs, extension pattern is more suitable, because in most cases host language’s programming environment may be extended to handle DSL constructs as well. Respectively, when DSL is not connected to any existing notation, more factors may support development of dedicated tool. However, some of advantages of extensions are independent from the DSL character, and makes this pattern suitable for most cases.

### 3.6 Summary

Domain-Specific Languages presents new approach to software development. Although the whole idea of using special purpose languages, which are focus on expressing terms from particular domain, is not new, DSLs became popular quite recently. Additionally, modern DSLs are more focused on modelling, rather than on implementation, thus most of them are graphical languages, where older are textual notations. DSLs may be built on top of existing language and exploit its advantages,
such as development tools or familiarity to host language users. When no language is fit for modifying it, DSL may be created from scratch, which makes DSL development more flexible, but is the source of additional problems. Costs of DSL development are most important disadvantage of Domain-Specific Development – development of new language is time consuming and difficult task, so before deciding in favour of DSL other options should be considered. However, advantages of DSLs such as reducing costs of software development or improving quality of software product outweigh disadvantages in most cases. The other problem of DSLs is the fact that DSL development process is poorly documented, and there is no precise definition of it. However, there were some well documented cases of successful DSL deployment. Based on those cases, there were attempts of structuring development process by dividing it into phases and identifying patterns associated with them. Nevertheless, DSL are still young concept, which will surely evolve during following years.
4 SharePoint platform

4.1 Introduction

Most of Enterprise Portals vendors ship their products in form of platform or framework, which is the base for building concrete portals. Working with each platform is different, because there is no standardized way of building them. Each EP vendor has its own vision of contents of such platform and way of working with it, and for that reason every portal developer is forced to follow those rules. In this chapter, the process of development of Enterprise Portals on SharePoint platform will be described in detail. This description is made in order to describe the process and identify problems, which occur during it.

- Section 4.2 provides brief introduction of SharePoint environment, by describing main products from SharePoint family and differences between them.
- Section 4.3 describes SharePoint architecture in terms of physical location of web sites and configuration databases, and explains basic SharePoint terminology.
- Section 4.4 presents principles of portal development on SharePoint platform. It describes various techniques and approaches to portal development on SharePoint platform, and how they are utilized during the development process. It also contains description of areas related to developments, such as platform’s architecture.
- Section 384.5 describes one of the most important aspects of development of portals based on the SharePoint platform, which is defining the structures for storing data. There are several data entities in SharePoint environment, each of which defines data structure on different level. This section provides definitions for all relevant elements of data model, along with description of method, by which they may be defined.
- In section 4.6, issues, which are associated with both development techniques and development artefacts are listed and described in detail. This section is crucial, because it describes problems which were solved by creating DSL. Their precise definition is necessary in order to verify if mechanisms proposed in the language actually solves them.
- Section 4.7 summarizes this chapter and provides table of development issues, along with their categorization.

4.2 SharePoint platform overview

One of the largest software vendors, Microsoft Corporation, has its own set of solutions designated to build Enterprise Portals, which is collectively known as SharePoint. SharePoint products and technologies, according to [21], provide capabilities to meet business-critical needs like managing content and business processes, simplifying managing (finding and sharing) information across enterprise boundaries and enable better informed decisions. The basis for all SharePoint technologies is Microsoft Windows SharePoint Services (WSS), currently in version 3.0. It is described as: “a foundation platform for building Web-based business applications that can flex and scale easily to meet the changing and growing needs of your business.” [24] WSS provides infrastructure for storing and managing various
kinds of data, tools which helps in collaboration, communication and project management, integrates various applications from Microsoft Office system, and enables business process automation. A higher-level product from SharePoint family is Microsoft Office SharePoint Server (MOSS) - a server program, which is “integrated location where employees can efficiently collaborate with team members, find organizational resources, search for experts and corporate information, manage content and workflow, and leverage business insight to make better-informed decisions.” [21] In other words, it provides functionality in areas of collaboration, portals, enterprise search and content management, business process automation, and business intelligence. MOSS relies on WSS technologies – is built on top of the WSS framework. However MOSS extends WSS by providing more complex components and services, which adds more value to the final products based on the SharePoint platform [22].

4.3 SharePoint architecture

From more technical perspective, “WSS is a site provisioning platform” [28], which enables constructing web sites, without manually creating new database and web pages, to store its order provide access to it. Physically, all SharePoint based applications are deployed within the farm, which is a set of one or more server computers, which provides functionality to clients (see Figure 5). The SharePoint farm consists of one database server, on which so called configuration database is running. Within this database, all necessary configuration information for particular farm is stored. The second layer of the farm consists of front-end Web servers, on which the actual SharePoint applications are running.

Figure 5. Example of WSS farm, based on [28]

WSS is built on top of Internet Information Services (IIS) – a web server, which includes a set of Internet-based services, such as handling FTP, HTTP/HTTPS, and SMTP. IIS web site is an entry point into the IIS Web server infrastructure, and is handling incoming HTTP requests. IIS Web site, which is configured for SharePoint
application is called web application [28]. Most important SharePoint entity is WSS site, which can be simply defined as “storage container for content.” [28] The data can be stored in many various containers, such as lists, document libraries, or child sites. Ability to store child sites within sites makes possible creating hierarchies of the sites, which is the one way of structuring content. Additionally, site is securable entity, which means that content stored within the WSS site may be accessible for configurable set of users. Finally, each site can be customized by changing its appearance, content and functionality in order to fit business needs associated with it [23]. Every WSS site runs within the context of web application, because it contains security configuration for user authentication. The second reason is the fact, that each web application is associated with content database - the storage for data from WSS sites. Sites cannot be directly associated with web applications. Instead, they are stored (along with their child sites) within site collections – containers for WSS sites, which are another way of simplifying administration and sharing common components. Each site collection must have its top-level (or root) site, and any site within the collection is the child of this site. Figure 6 illustrates simple example of relations between web application, site collections and site hierarchies.

![SharePoint entities hierarchy](http://www.litwareinc.com:1000)

**Figure 6. SharePoint entities hierarchy, from [28]**

### 4.4 SharePoint development techniques

As a portal platform, SharePoint environment allows several approaches to making portals, which differs in terms of developers’ engagement and level of abstraction, on which portal components are defined and configured. They can be also distinguished considering methods, which are used to operate SharePoint environment. Those methods include customizing particular instance of portal (customization), working with portals and their definitions programmatically (via object model) and stating portal contents declaratively (features and site definitions). All those methods are described in detail in following sections.
4.4.1 Customization versus Development

In the SharePoint environment, there is a clear differentiation between customization and development. The customization is done by introducing changes via web browser or tools such as Microsoft Office SharePoint Designer (SPD) and those changes are recorded by modifying data in SharePoint content database. In literature, the result of customization is called customized content [28]. The main advantage of customization is the fact that it can be done by users or administrators, who do not have a programming skills. Additionally, customization is usually quick and can be performed ad-hoc, so its effects are visible almost immediately. Although customization in SharePoint is quite flexible, it has its limits and quite often business needs cannot be satisfied with customization alone. Another liability of customization is the lack of mechanism allowing versioning and repeatability of introducing changes, thus it is quite hard to move customized content between development and production environment.

The development on SharePoint platform provides more opportunities than the customization, mostly because it is performed beneath the level of customization. Development is associated with working with source files (such as XML definitions, compiled assemblies etc.), which are often called provisioning components, and deploying them on the front-end Web server [28]. The fact that the changes are done outside the content database, enables more professional development approach, which includes centralized code management (versioning and integration) and reusing provisioning components across multiple sites.

4.4.2 SharePoint object model and Web Parts

SharePoint is an application built on top of the ASP.NET 2.0 and .NET 3.0 frameworks. On the other hand, it is an application framework on its own – it provides managed API [10], which can be utilized by developers in order to benefit from full potential of the platform. The standard WSS assembly Microsoft.SharePoint.dll contains rich set of class and methods, allowing interaction with SharePoint environment at level of code. The main functionality which is enabled by WSS object model is associated with working with data. It allows modifying behaviour of the various SharePoint entities (such as site collections or sites), creating and modifying new data structures (content types, lists), operating on data items stored in those structures. Additionally, object model enables defining event handlers which provide additional business logic when some events (such as adding or modifying items, activating features etc.) occur. For example, one of the most important classes, SPSite, represents SharePoint site collection, and serves as the entry point to the rest of SharePoint environment. It has a property, which contains the top-level site of the site collection (an object of SPWeb class), and the collection of all sites within site collection. SPWeb class has collection of lists, which are present on the particular web site. Listing 1 presents an example of working with SharePoint object model.
Another important aspect of developing SharePoint solutions is building Web Parts. Basically, Web Parts are rich ASP.NET server controls, which can be managed directly from user interface (UI) by end users [27]. Web Part management include its adding or removal from particular site, changing its position within a site, but most important aspects of those controls are customization and personalization. Customization of Web Part is to make changes to some of its properties (such as data source or theme of layout). Personalization means, that changes made by particular user will be visible only for him, they will not affect Web Parts of the same kind viewed by others. Web Parts can be considered as equivalent of portlets. Web Parts in SharePoint environment are key server side controls that allow various kinds of actions from managing data in lists, through performing search and navigating throughout site collections, to actions as sophisticated as generating reports or performing administering tasks. Encapsulation of functionality, and ability to customize many of Web Part properties enable its reusing in many SharePoint sites, thus making them valuable assets in building complex SharePoint solutions.

4.4.3 Features

Features are a development-oriented mechanism, which allows defining and adding site elements to desired site or site collection by installing and activating them. They can be viewed as pieces of functionality, which can be easily installed, turned on and off. The advantage of using features is the ability to reuse same functionality across multiple sites, by activating feature on each of them, without re-creating this functionality by copying large pieces of XML code. At the physical level, feature is a directory within system directory, which is located on each front-end Web server, on which WSS is installed. This directory contains file named feature.xml – a definition of the feature written in Collaborative Application Mark-up Language (CAML). CAML is XML-based language designed for SharePoint. The feature definition consists of basic properties of the feature (its ID, title, description etc.), and lists elements bound to it, such as manifests of files associated with feature, or custom

```csharp
class WSSObjectModelSample
{
    static void Main()
    {
        string siteUrl = "http://example.com";
        // acquiring object of site collection having its url
        SPSite site = new SPSite(siteUrl);
        // acquiring object of root site in site collection
        SPWeb web = site.RootWeb;
        // enumerating the lists within site
        foreach (SPList list in web.Lists)
        {
            Console.WriteLine(list.Title);
        }
        // disposing objects
        web.Dispose();
        site.Dispose();
    }
}
```

Listing 1. Example of working with SharePoint object model
feature properties. Manifest of elements contained in feature is usually stored in separate file.

There are also many advanced methods for working with features. They have such properties as Hidden, which determine if the feature can be activated and deactivated via user interface, and Scope, which determines whether changes made by activating features affects single site (Web), or whole site collection (Site). The activation dependency mechanism is used to express a requirement in the relationship between two features [23]. It is used for two purposes: for grouping features, when their activation provides more sophisticated business functionality, and for ensuring resources necessary for correct feature activation. Example of contents of feature.xml file is shown on Listing 2.

```xml
<Feature
  Id="{19212403-EA69-4b91-80EC-7E1747D7CEB5}"
  Title="Custom list instances"
  Description="This feature installs custom list instances"
  Version="1.0.0.0"
  Scope="Web"
  Hidden="FALSE"
  ImageUrl="menuprofile.gif"
  ReceiverAssembly="Samples, Version=1.0.0.0, Culture=neutral, PublicKeyToken=b59ad8f489c4a334"
  ReceiverClass="Samples.FeatureReciever"
  xmlns="http://schemas.microsoft.com/sharepoint/">
  <ElementManifests>
    <ElementManifest Location="elements.xml" />
  </ElementManifests>

  <ActivationDependencies>
    <!-- Feature that installs custom list templates -->
    <ActivationDependency
      FeatureId="{A032B7DB-38C0-4b99-A110-D7BFBE7B23FF}"/>
  </ActivationDependencies>
</Feature>
```

Listing 2. Example of feature definition

For example, feature called “My Favourite Items”, could provide functionality supported by: custom list, which stores particular user’s favourite items, a custom menu item “Add to Favourite”, which adds reference to particular item on the list of favourites etc. [23]. This feature could depend on another feature, which deploys template of favourites list.

4.4.4 Site definitions

More business-oriented package of functionality in SharePoint environment is called site definition. Each site that exists within the SharePoint environment is created from a site definition. Site definition is a “top-level component in WSS that aggregates smaller, more modular definitions to create a complete site template that can be used to provision sites.” [28] Those “modular definitions” include custom list definitions, Web Parts, pages or features. Site definitions provide ability to combine standard SharePoint functionality with custom components, and providing pre-
packaged business solutions as a result. Sites are dependent on its definition through its whole lifetime, thus the definition for a provisioned site cannot be changed or removed. Even if feature is activated within the site, its definition remains intact, so the next site provisioned from that definition would not include changes made by this feature. At the physical level, site definition is very similar to a feature – it is a set of CAML-based configuration files, custom pages and controls (.aspx, .ascx files), and document templates (.doc, .xls, .html) which all are deployed within dedicated system directory on front-end Web server. WSS is shipped with several built-in site definitions, which are good examples of how this kind of business solutions should be built. For example, team site definition consists of elements such as list of team members, team calendar, and message board.

### 4.5 SharePoint data model

Many definitions of the Enterprise Portals, such as those provided by [32] or [38], states that one of the most crucial aspects of the EP is the way that the content is stored within it. The content may include documents, information about people and other entities and even metadata, which describes the actual data [28]. As the EP platform, SharePoint contains flexible data model, which allows defining the data on several different levels. Those levels are represented by SharePoint entities such as lists, content types, or columns, which will be described in following subsections. The other common property of those entities is the method of creating them, which can be achieved in three ways [23]:

- By using WSS user interface
- By using WSS object model
- By deploying a Feature with XML definition file.

Defining data structures by using WSS user interface is the easiest way, however it enables reusing those structures in limited way, because changes made via UI are customization changes, thus they are stored in content database. The outcomes of the following methods are provisioning components, so definitions made that way are flexible and reusable. Assemblies made by building components via object model can be easily deployed on servers, as well as directories containing features and files associated with them.

#### 4.5.1 Columns

Column (or site column), can be defined as reusable definition or template, which “represents an attribute, or piece of metadata, that the user wants to manage for the items in the list or content type to which they added the column.” [23] The column is closely related to the attribute (or column) from the relational database theory [5], thus it is defined by its name and value set (data type). In WSS object model and CAML, columns are commonly referred as fields, thus in this work both of those terms will be used.

The idea that stands behind the column is to decrease re-work associated with defining similar attributes for different data entities. For example, once the columns such as Name or Personal ID number are defined, they can be included in multiple
lists or content types. Additionally, once the change is made to the column definition, those changes will be applied in all entities which include this column. However, propagation of those changes may be broken if necessary. There are several different methods defining columns, as well as ways of reusing them.

As it was mentioned before, column basic data types in WSS include single or multiple lines of text, number, currency, date and time. Additionally, some detailed settings associated with chosen data type can be specified, as well as the group, to which column belongs. Grouping is very useful mechanism of organizing columns into logical categories, and is leveraged in SharePoint environment in many situations, such as adding columns to lists and content types via UI. Another important property of column is its scope – the area where particular column can be used. Columns defined as site columns are available in every list and content type within children of the site. Once some common columns are defined, they can be reused in many places. On the other hand, column can be defined as local list column, thus it can be used only within that list. This allows defining specific columns only in places, where it is needed.

Listing 3 presents an example definition of column. The column, which name is Gender is based on underlying Choice field type. The way the options will be visible in user interface is defined in Format attribute (list of radio buttons in this case). Additionally, two possible choices are defined, along with the default value, and the group, which will contain the column. The ID attribute is a Globally Unique Identifier (GUID), which is additional way of identifying the column within the SharePoint environment.

```xml
<Field
    ID='{0861685F-025C-417b-90D4-0B1F641CA02A}'
    Name="Gender"
    DisplayName="Gender"
    Type="Choice"
    Format="RadioButtons"
    Group="Employee Columns">
    <CHOICES>
        <CHOICE>Male</CHOICE>
        <CHOICE>Female</CHOICE>
    </CHOICES>
    <Default>Male</Default>
</Field>
```

**Listing 3. Example of XML definition of the column**

Listing 3 presents an example definition of column. The column, which name is Gender is based on underlying Choice field type. The way the options will be visible in user interface is defined in Format attribute (list of radio buttons in this case). Additionally, two possible choices are defined, along with the default value, and the group, which will contain the column. The ID attribute is a Globally Unique Identifier (GUID), which is additional way of identifying the column within the SharePoint environment.

### 4.5.2 Field types

Custom field types are necessary when WSS field types are not sufficient for storing more complex types of data, or they require some enhancements to fit into business needs. It is possible to acquire more control over initialization, rendering and validation of filed, which is based on custom field type [28]. In order to create custom field type, several tasks must be completed [23]. First task is to create custom field type definition, which basically is CAML-based XML file containing information necessary to properly render fields in WSS forms, and the reference to assembly with compiled field class, which writing is the second necessary step. Field class inherits
WSS SPField class, or one of the classes that derive from it. It contains additional rendering instructions, properties and data validation logic. Additional entities that may be used during creating custom field types are: rendering class and template, which enables advanced field rendering), custom value class or structure for storing field data, and editing control for the variable properties of the field type, which enables defining properties relevant for the field of custom field type.

4.5.3 Content types

Content type is another way of organizing content within SharePoint in a more manageable, meaningful and reusable way. Content types are defined as: “reusable collection of settings you want to apply to a certain category of content.” [23] Those settings include the set of columns and behaviour for items (in lists) or documents (in document libraries), thus they provide means for encapsulating data requirements [28]. The closest analogue for content type from relation model for database is relation schema [5].

![Listing 4. Example of XML definition of a content type]

The main function of content types is to define schema for items and documents. The fact, that content types are defined outside the scope of lists and document libraries allows reusing them in multiple, more complex data storage entities, in the similar way as columns. This approach enables content standardization – once the entity such as employee is defined as a content type, every list, which is supposed to store information about employees can do it by describing this entity with the same set of attributes (columns of the content type). Similar approach can be applied to documents - in order to standardize their design and content, document templates may be created and associated with content types. One of the most important advantages of content types is the fact that all content types are built based on the principles of inheritance [28]. This allows creating more general content types, which stores common set of columns and serves as the base content type for more specific ones. If there are some special kinds of employees (e.g. temporary employees), the appropriate
content type can derive from employee content type, by enhancing it with additional attributes. Next benefit of content types, in case of associating them with documents, is their independence from the file formats. It is possible to associate content type with PDF or Word documents, spread sheets, pictures, videos and any other kind of file. It is also possible to define content type, which is not associated with any kind of file. For this reason, content types can be categorized into two groups: document-based and item-based. There is also another way of categorizing content types, which is based on their scope. Usually, content types are defined at the level of site (site content types), but when one of them is assigned to a list, its local copy is made within the list (list content type). This allows modifying content type on the one, particular list without applying those changes to other lists. Listing 4 presents an example of XML-based content type definition.

**Listing 4**

**Figure 7. Different kinds of content types and content type inheritance** [23]
Figure 7 presents the site and list content types along with example of content type inheritance. Both Spec and Memo site content types derives from Document, which is one of the base SharePoint content types. The ProductXSpec is site content type defined in child site, and Spec is its base content type. There are also ProductXSpec and Memo content types, which are the list content types.

4.5.4 Lists and document libraries

Lists and document libraries are one of the most important concepts when talking about SharePoint. Most of the work associated with both setting up SharePoint applications and using them in business environment, consist of defining list schemas, creating list instances and operating on list items. Basically, lists and document libraries are actual storages for data. User, by creating choosing content type, and fills particular columns with necessary data, is creating new item on the list. Similarly, by uploading document and describing its metadata, user is creating new item in document library. Lists instances are SharePoint equivalents of database tables [5].

WSS is shipped with several built-in list types, which provides a basic functionality sufficient in many business scenarios [28]. Those types include document library, which used to store documents, and supports versioning, workflow, and checking documents out (blocking in order to make some changes) or in (returning them into library). Similar to document library type of list is called form library – it is used to store XML documents and forms, which can be used with Microsoft Office InfoPath. Calendar list is used for tracking events and deadlines, and can integrated with calendar in Microsoft Office Outlook, task list is used to store activity-based items (such as workflow tasks), and project tasks is task list enhanced with Gantt chart view and integration with Microsoft Office Project. Those types, along with several unmentioned in this work, can be basis for creating custom lists definition, which are more adapted to specific business needs. List types and definitions are only templates for actual lists, also known as list instances. There may be several list instances built from the same list definition, so list definition is another way of reusing data structures. Thanks to that, information, which has the same schema (information about employee), can be partitioned among many entities (list of IT department employees, list of management department employees – all based on the same list definition - list of employees).

One of the most important properties of the list is the fact, that there can be multiple content types associated with one list. This enables storing heterogeneous data in one place, which makes SharePoint data storage model even more flexible. For example, there could be a common document library, which supports multiple content types, where all documents that arrived in company are stored temporarily, until business worker move them to specific document libraries, where documents of only one type can be stored.

List definition is considered to be most complex XML-based definition in SharePoint, because of its length – usually several thousand lines [28]. The size of the list definition is determined by the fact that it contains many sections such as content type references, local definition of all columns that are incorporated into list, forms and most of all – views, which contains CAML rendering instructions. Additionally, there is also increased complexity in physical structure of the list definition. When list
is to be deployed via feature, the manifest of the feature should, as usual, contain
reference to the file with elements of that feature. In this file (Listing 5), list template
element should be specified, along with its attributes such as Name and
DisplayName, Type and BaseType, attributes which specifies whether list supports
versioning and attachments etc.

```
<ListTemplate
  Name="EmployeesList"
  Type="10001"
  BaseType="0"
  DisableAttachments="FALSE"
  VersioningEnabled="TRUE"
  Hidden="FALSE"
  Sequence="2000"
  DisplayName="Employees"
  Description="Create a custom employees list."
  Image="/_layouts/images/itcontct.gif"
  OnQuickLaunch="FALSE"
 />
```

**Listing 5. Example of XML definition of list template**

Name of the list is crucial attribute, because it determines name of the directory
within feature directory, which contains schema.xml — file, which contains
definition for list schema (Listing 6). This definition enables overwriting some of the
attributes specified in list template, as well as defining list metadata, which is most
important part of the list, because it contains, as mentioned before, references to
content types, fields, views and forms.

```
<List
  Title="Employees"
  Url="Lists/Employees"
  BaseType="0"
  DisableAttachments="TRUE"
  VersioningEnabled="TRUE"
  EnableContentTypes="TRUE"
  FolderCreation="TRUE"
  xmlns="http://schemas.microsoft.com/sharepoint/"
>
  <MetaData>
    <ContentTypes> <!-- Section for content types -->
    </ContentTypes>
    <Fields> <!-- Section for fields --></Fields>
    <Views> <!-- Section for views --></Views>
    <Forms> <!-- Section for forms --></Forms>
  </MetaData>
</List>
```

**Listing 6. Example of XML definition of a list schema**

The next and final step in defining list is via XML files is instantiating the list. This
step is not necessary, because having properly installed list template, it is possible to
create list instance from this template via user interface. Nevertheless, there are cases,
when business scenario requires hiding list template (Hidden="TRUE") and creating
only one instance of it. In that case, instance can be defined in CAML (Listing 7) and
deployed via feature activation. TemplateType attribute of ListInstance
element corresponds with the Type attribute of ListTemplate, and it is an indicator of the template on which instance is based. Again, some of the list template properties can be overridden (OnQuickLaunch), but those changes will apply only to one particular instance. List instances are also useful, because they enable populating lists with data, which is very useful for testing. Within Data node, multiple Row elements can be specified, where each of them represents item on the list, and is defined assigning values to each of its columns, which are identified by their names. The ContentType field is an internal WSS field, which is necessary for identifying the content type for particular item.

```
<ListInstance
    TemplateType="10001"
    Id="Vendors"
    Title="Employees"
    Url="Employees"
    Description="List of employees."
    OnQuickLaunch="TRUE">
    <Data>
        <Rows>
            <Row>
                <Field Name="ContentType">Employee</Field>
                <Field Name="Title">Doe, John</Field>
                <Field Name="FirstName">John</Field>
                <Field Name="FullName">Doe</Field>
                <Field Name="Gender">Male</Field>
                <Field Name="WorkPhone">(813)345-5432</Field>
                <Field Name="JobTitle">Manager</Field>
            </Row>
        </Rows>
    </Data>
</ListInstance>
```

Listing 7. Example of XML definition of list instance

### 4.5.5 Views

Views are important aspects related to lists. SharePoint views are basically the same as database views [5] - they represent virtual list or document library, in which elements are stored in manner defined in query. Query consists of filter, which defines conditions for list elements to be displayed, order in which those elements appear and maximum number of elements in the view. Apart from that, there must be also definition of the fields that are to be displayed in view. Views, as it was mentioned in section 4.5.4, are defined as part of list definition, in schema.xml file, and they are one of the main factors determining the enormous size of those files. The reasons are rendering instructions, which are part of every view definition. Listing 8 presents an example of view definition, where first nine nodes (GroupByHeader, GroupByFooter, etc.) contain rendering instructions for various view sections (header, footer), and those sections are usually exactly the same in different views, so they may be copied from views shipped with WSS. Following nodes (ViewFields, Query, RowLimit) are more view-specific, and have to be specified manually.
The most flexible mechanism for development of provisioning components is to define them declaratively in XML files using Collaborative Application Mark-up Language. After that, those components may be deployed using features or site definitions. Although this method provides agility and reusability much wider than others (customization or working with object model), there are many issues associated with it. Those problems are usually caused by large XML files, which are difficult to manage and read. There are class of problems, which are associated with defining relationships between various SharePoint entities. Those relationships are defined by referencing entities by their unique identifiers, which is uncomfortable for developers. There is also lack of methods for visualising those relationships, which is major obstacle for portal architects. In following sections issues associated with SharePoint development are discussed in details.

4.6 Issues in development process

The most flexible mechanism for development of provisioning components is to define them declaratively in XML files using Collaborative Application Mark-up Language. After that, those components may be deployed using features or site definitions. Although this method provides agility and reusability much wider than others (customization or working with object model), there are many issues associated with it. Those problems are usually caused by large XML files, which are difficult to manage and read. There are class of problems, which are associated with defining relationships between various SharePoint entities. Those relationships are defined by referencing entities by their unique identifiers, which is uncomfortable for developers. There is also lack of methods for visualising those relationships, which is major obstacle for portal architects. In following sections issues associated with SharePoint development are discussed in details.

4.6.1 Defining columns

The numerous attributes of Field element enables precise definition of the column. For example when the data type for the column is Choice (Listing 3), it is possible to define other properties associated with this data type (such as method of rendering in UI or set of values that are available to choose) by assign values to Field attributes or its child elements.

Listing 8. Example of XML definition of a view

```xml
<Views>
  <View
    BaseViewID="2"
    Type="HTML"
    WebPartZoneID="Main"
    DisplayName="My Tasks"
    MobileView="TRUE"
    MobileDefaultView="TRUE"
    SetupPath="pages\viewpage.aspx"
    ImageUrl="/_layouts/images/issues.png"
    Url="MyItems.aspx"
    ReqAuth="TRUE">
    <GroupByHeader />
    <GroupByFooter />
    <ViewHeader />
    <ViewBody />
    <ViewFooter />
    <PagedRowset />
    <PagedClientCallbackRowset />
    <PagedRecurrenceRowset />
    <ViewEmpty />
    <ToolBar Type="Standard" />
    <ViewFields> <!-- Section for view fields --> </ViewFields>
    <Query>
      <Where> <!-- Section for query condition --> </Where>
      <OrderBy> <!-- Section for query ordering --> </OrderBy>
    </Query>
    <RowLimit Paged="TRUE">100</RowLimit>
  </View>
</Views>
```
The fact, that most of the properties depends on the chosen data type is weakness of this method of defining columns. Columns schemas differentiate a lot, and lack of mechanism for static analysis and validation of column definition makes determining, if all necessary attributes has been specified, very difficult. Additionally, inexperienced developers, whose knowledge and experience with CAML schema is limited, may have difficulties in identifying those attributes, which applies to chosen data type. Column definitions made by them may not take full advantage of flexibility provided by this method of creating columns. There is also case, when data type of the column is a custom field type – then there is no documentation support for identifying those properties, which assigning is necessary for the column to work properly. Visualisation of the relationships (inclusion) between columns and groups would be helpful in verifying if all necessary columns are defined and if they are members of right group.

4.6.2 Defining Lookup fields

Among many data types in WSS, Lookup field type is extremely important. A field, which type is Lookup can reference a field value in another list within site, so it makes possible to define relationships between items on different lists. The role of the lookup field is similar to the role of foreign key in database [5].

Lookup field is very difficult to define during design phase, because it requires the GUID of the target list, as well as GUID of the target site and internal name of the field, which is referenced by lookup. The GUID of the list is not deterministic – each time the list instance is created, its ID is assigned by SharePoint. When the list and field are designed simultaneously, and will be deployed and instantiated within same solution, specifying all attributes is impossible. This brings out the need for mechanisms that would automatically fill missing values in definition, basing on information contained in model. There is also need for notation, which would illustrate relationships between lists in more convenient way, rather than as the IDs of the list within the lookup field definition.

4.6.3 Content type IDs and inheritance

The content types have very sophisticated identifiers, which “encapsulates the line of the parent content types from which the content type inherits.” [23] This allows determining how the two content types are related. There are two conventions of creating IDs for new content types: by adding two hexadecimal digits to the parent content type ID, or by adding “00” and hexadecimal GUID. The example of content type hierarchy and appropriate IDs is shown on Figure 8.
There are several issues associated with the content type ID. The first is the way of creating it, which requires the knowledge about parent content type ID. This seems to be easy, when deriving from the basic SharePoint content types (such as Item or Document), but while the line of inheritance became longer, the content types IDs became more complex. Additionally, the analysis of content type ID is the only way of determining the relationship between content types. Parsing the content type ID and comparing its components with IDs of other content types is inadequate way of describing relationships – the visual representation would be more convenient and readable.

4.6.4 Referencing columns in content types

One of the most fundamental tasks in defining data structures such as list or content types is choosing which columns will be included in those structures. When defining content type in XML file, set of content type columns must be referenced in appropriate section of the definition (Listing 4). The fact, that columns are referenced allows including only existing or already defined columns – it is not possible to create local column within content type definition. Figure 9 presents the relation between site columns and content types. The ID of the field reference must retain a GUID of existing content type in order to be valid.

The fact, that field reference ID – the identifier of the column in content type definition – is a GUID, brings some difficulties in defining content types. GUIDs are not readable and easily verifiable, so usually each field reference contains also name of the column. Nevertheless, the ID is decisive and it must be correct. The other issue is associated with the content type inheritance. References to the columns are inherited from parent content type in child content types, so they do not have to be specified again. However, when the line of inheritance is long, the developers may be confused about fields in content type, and some references may be added again, causing errors during content type deployment. There is also problem in identifying the content type.
types, which contains particular column – the only way to do that (at the level of XML definitions) is to examine field references of all content types.

Figure 9. Columns in content types and lists, from [23]

4.6.5 Adding columns to lists

In contrast to content type, list stores its own local definitions of each column that is added it. The list definition contains section, where fields may be defined once again. If list column was created from site column, the ID of list column should correspond with ID of site column. This consistency enables propagating changes made in site column in each list column, which is based or ‘inherits’ it. There is also possibility to create list column without site equivalent. The relation between list columns and site columns is shown on Figure 9.

The problem with adding columns to list is similar to the problem in referencing columns in content types, although it is more complex. Apart from the consistency problem, which basically is the same (ID in the list must be the same as ID of existing column, unless new column is to be created), the necessity of defining columns once again is a source of additional work. Additionally, this work boils down to copying and pasting field definitions, so there is possibility of automating it.

4.6.6 Adding content types to lists

Adding columns to list is done mainly in order to add content type to the list. It is necessary every time, when additional content type is to be added to list. The only exception is the situation, when the column of content type is already defined in list. There is also possibility to extend local list content type with few additional columns, which are defined within the list. The content type can be added to a list in two ways – first is the referencing content type by adding ContentTypeRef node in content types section. This element contains only the ID of the existing site content type. Another way is to create a content type within a list from scratch (ContentType node) – in that case all necessary content type attributes and child elements must be
specified. Relationships between site- and list-scoped content types are shown on Figure 10.

Each of methods for adding content type to a list has its advantages, yet all of them require operating on IDs of existing site content types. When creating content type reference, its ID must correspond with ID of site content type. When creating content type from scratch, its ID must be valid in terms of content type inheritance. No matter which method is chosen, there is also necessity of adding appropriate columns to list definition. When content types are added to list via object model, WSS automatically adds the necessary fields from the content type into Fields collection of the list. Unfortunately, when working with XML definitions, each field must be explicitly defined once again, by referencing its ID, and including other attributes such as Name, Type etc. This is, like in previous case, source of work, which can be fully automated. There is also risk of defining the same column more than once, for example when parent and child content types are added to list - in such case, list definition is invalid. The last issue is associated with identifying the content types that are part of more than one list – the only way to determine that is to examine appropriate section of lists in search for ID of content type.

### Figure 10. Site-scoped and list-scoped content types and columns, from [23]

#### 4.6.7 Defining views

As it was stated in section 4.5.5, views are basically parts of list definition. The more views are associated with the list, the more complex and larger the list definition
became. In most business cases, there are several views associated with list, so large size of file containing list definition is unavoidable.

The necessity of working with views directly in list definition is problematic, because it requires working with XML file, which size is measured in thousands line of code. In order to create view, the rendering instructions must be copied from already existing, pasted under appropriate node of list definition, and customized by defining the view name, columns, query etc. There is also necessity of verifying, if columns listed in view are defined in list, or else the whole list definition will be invalid. The most difficult activity during modifying view is finding it. After that, activities analogous to those from creating view may be performed. The other problem is lack of methods for showing all views associated with list and comparing them, which would be useful in verifying if all necessary views are defined and if there are no views, which are redundant.

4.6.8 Defining features

There are some rules for activation dependencies, which assists avoiding circular dependencies, dependencies chains that limit performance etc. There are three groups of rules: general, same-scope and cross-scope. General rules applies to all activation dependencies, same-scope rules – to the features, which scope is the same, and cross-scope rules – when feature which is dependant has scope different than feature on which it depends.

Lack of mechanisms for visualising activation dependencies between features makes their validation difficult. In large projects, when there are lot of features and activation dependencies, finding the features on which particular feature depends and checking if those dependencies are valid, which in most cases means that activation dependencies of the dependant features must be examined as well, is very demanding activity. When some of the rules are not abide, some of the features may end up in invalid state. Additionally, defining feature dependencies in XML files is not suitable for portal architects, thus making this mechanism inadequate for deployment planning. There is also another problem associated with defining content of the feature. All entities which are to be installed by the features (such as columns, content types, lists, etc.) must be manually listed in feature definition. Those references ought to be created automatically, in order to save developers time. Additionally, association between feature and its elements should be presented graphically, which would help portal architects in grouping portals elements depending on their function and fulfilment business needs.

4.6.9 Defining sites

Sites cannot be created directly from site definitions. Instead, site definition contains zero or more configurations that group functionality. Site definition that contains no configuration is abstract and cannot be instantiated, yet it can serve as storage for common infrastructure definition, which can be reused in other site definitions. Based on configurations, actual site templates are defined.

Problems with site definitions are similar to those, considering list definitions or feature definitions. The main issue is complex structure of site definitions, which is spread among two files. First of them is onet.xml (see Listing 9), which contains
navigation bars (NavBars), templates for documents that will be used in document libraries (DocumentTemplates), and modules that define files, that are to be deployed along with SharePoint site (for example .aspx files). Most important part of onet.xml is section, in which configurations are defined. This file is content of site template directory, which is stored within SiteTemplates directory - special folder on front-end server. Any kind of element listed in configuration is referenced only by ID, so identifying it is source of additional work. Creating and modifying site definition is also complex task, because in most cases, features should be listed in specific order, which is the order in which they will be activated. Visualisation of associations between site definition and its content would decrease complexity of creating and managing site definitions. Automatic code generation, which would generate references to site definition content, would be also helpful.

```xml
<NavBars>
  <!-- Navigation bars -->
</NavBars>
<DocumentTemplates>
  <!-- Templates for documents -->
</DocumentTemplates>
<Configurations>
  <Configuration Name="ExampleSiteTemplate" ID="0">
    <Lists>
      <!-- Lists -->
    </Lists>
    <Modules>
      <!-- References to modules -->
    </Modules>
    <SiteFeatures>
      <!-- Site-scoped features -->
    </SiteFeatures>
    <WebFeatures>
      <!-- Web-scoped features -->
    </WebFeatures>
  </Configuration>
</Configurations>
<Modules>
  <!-- Definitions of modules -->
</Modules>
```

Listing 9. Example of site definition stored in onet.xml file

The second kind file is called webtemp*.xml file, from the specific naming convention (* is the name of site definition). Those files are stored also on front-end server, in directory which is named corresponds with locale identifier (1033 for US English, 1045 for Polish etc.). Those files contain definitions for templates, which consist of reference to particular site configuration (Listing 10). In order to make configuration visible for users via WSS UI Value of template attribute SetupPath must correspond with name of the directory, which contains onet.xml file with configurations definitions. Likewise, ID of configuration must be one of IDs specified in onet.xml. Automatic template generation would eliminate risk of specifying invalid values for described attributes.
When working with SharePoint object model, access to various SharePoint entities follows the same pattern. As it was stated before, within SharePoint object model exist many classes, which instances represents various SharePoint entities, such as SPSite for site collection, SPWeb for site, SPList for list, and SPContentType for content type to name a few. Those objects contain methods and properties, which provide access to objects contained within them. For example (Listing 11), SPList object has Items property, which returns SPListItemCollection object. This object contains collection of SPListItem objects, and represents all items stored within particular list. There is also possibility to gain access to item directly from SPList object, by invoking methods such as GetItemById or GetItemByUniqueId. Access to other SharePoint entities from different objects can be gained in similar way.

There are several ways to acquire concrete item from collection, which depends on the type of the value specified in collection’s indexer. When trying to obtain particular content type from list, it is possible use ContentTypes property of SPList object, and then specifies its index within collection, identifier or its name. It is also possible to invoke methods such as BestMatch, which would also result in acquiring SPContentType object. Those mechanisms are quite flexible, because they provide means to access particular item having different data associated with it. The real problem is to acquire that information, which identifies particular item in collection. The item’s index is bound more to the collection than to the item itself, thus this method is used very rarely. Identifiers of content types are, as it was stated in section 4.6.3, complex and difficult to interpret, so access via content type name seem to be most convenient. However, spelling errors, and possibility to change content type name at runtime (which is impossible for content type ID), proves that this approach is not without frauds. SharePoint itself provides some means, which solves those problems in limited scope. There are classes such as SPBuiltInFieldId or SPBuiltInContentTypeId, which retrieves values of, accordingly, GUIDs for columns and content type IDs for content types. Unfortunately, those classes contain identifiers only for built-in columns and content types. Nevertheless, there is a strong need for similar solution, which would also include metadata of custom columns and content types. Additionally, such storages for identifiers for other SharePoint entities, such as list templates, may be useful.

Listing 10. Example of template definition stored in webtemp*.xml file

```xml
<Templates xmlns:ows="Microsoft SharePoint">
  <Template Name="ExampleSiteTemplate" ID="34013"
    SetupPath="SiteTemplates\ExampleSiteDefinition">
    <Configuration ID="0" Title="Example of site Template"
      ImageUrl="/_layouts/images/stsprev.png"
      Description="Site template description"
      DisplayCategory="Site template category">
    </Configuration>
  </Template>
</Templates>
```

4.6.10 Working with SharePoint collections

When working with SharePoint object model, access to various SharePoint entities follows the same pattern. As it was stated before, within SharePoint object model exist many classes, which instances represents various SharePoint entities, such as SPSite for site collection, SPWeb for site, SPList for list, and SPContentType for content type to name a few. Those objects contain methods and properties, which provide access to objects contained within them. For example (Listing 11), SPList object has Items property, which returns SPListItemCollection object. This object contains collection of SPListItem objects, and represents all items stored within particular list. There is also possibility to gain access to item directly from SPList object, by invoking methods such as GetItemById or GetItemByUniqueId. Access to other SharePoint entities from different objects can be gained in similar way.

There are several ways to acquire concrete item from collection, which depends on the type of the value specified in collection’s indexer. When trying to obtain particular content type from list, it is possible use ContentTypes property of SPList object, and then specifies its index within collection, identifier or its name. It is also possible to invoke methods such as BestMatch, which would also result in acquiring SPContentType object. Those mechanisms are quite flexible, because they provide means to access particular item having different data associated with it. The real problem is to acquire that information, which identifies particular item in collection. The item’s index is bound more to the collection than to the item itself, thus this method is used very rarely. Identifiers of content types are, as it was stated in section 4.6.3, complex and difficult to interpret, so access via content type name seem to be most convenient. However, spelling errors, and possibility to change content type name at runtime (which is impossible for content type ID), proves that this approach is not without frauds. SharePoint itself provides some means, which solves those problems in limited scope. There are classes such as SPBuiltInFieldId or SPBuiltInContentTypeId, which retrieves values of, accordingly, GUIDs for columns and content type IDs for content types. Unfortunately, those classes contain identifiers only for built-in columns and content types. Nevertheless, there is a strong need for similar solution, which would also include metadata of custom columns and content types. Additionally, such storages for identifiers for other SharePoint entities, such as list templates, may be useful.
SharePoint, like all other Enterprise Portal platforms, is unique and specific. SharePoint provides its own set of terms, which are associated with various aspects of the environment such as its physical structure or architecture. Understanding of those terms is necessary for portal development. SharePoint enables several different approaches to Enterprise Portal development, from simple customization via UI to using API that is provided by the environment. Custom development is the most flexible and most complex technique of developing portals. There is a strong need for improvements, which would reduce the time and cost of development. Improvements may be made by eliminating problems that occur during development. However, before suggesting solutions, problems must be properly identified and described. Table 6 lists and categorizes all issues associated with the development process presented in section 4.6. Each problem is briefly described from three different perspectives: Code generation, Validation, Visualisation. Code generation problems are associated with the necessity of performing repetitive programming tasks and may be solved by

```csharp
class WSSCollectionsSample
{
    static void Main()
    {
        string siteUrl = "http://example.com";
        SPSite site = new SPSite(siteUrl);
        SPWeb web = site.RootWeb;

        // acquiring list, which name is Employees
        SPList employeesList = web.Lists["Employees"];

        // Id of Temporary Employee content type
        SPContentTypeId temporaryEmployeeCTId = new
            SPContentTypeId("enter CT Id here");

        // Id of Contract End Date column
        Guid contractEndDateId = new Guid("enter Id here");

        // Id of Gender column
        Guid genderId = new Guid("enter Id here");

        // display name of Last Name field
        // WARNING: it may be modified!
        string lastNameDN = "Last Name";

        // checking condition for every item on list
        foreach (SPListItem item in employeesList.Items)
        {
            if (item.ContentType.Id == temporaryEmployeeCTId)
            {
                string message = String.Format("Employee {0} ends {1} contract on {2}",
                    item[lastNameDN].ToString(),
                    item[genderId].ToString() == "Male" ? "his" : "her",
                    item[contractEndDateId].ToString());
                Console.WriteLine(message);
            }
        }
    }
}

Listing 11. Example of working with SharePoint collections

## 4.7 Summary

SharePoint, like all other Enterprise Portal platforms, is unique and specific. SharePoint provides its own set of terms, which are associated with various aspects of the environment such as its physical structure or architecture. Understanding of those terms is necessary for portal development. SharePoint enables several different approaches to Enterprise Portal development, from simple customization via UI to using API that is provided by the environment. Custom development is the most flexible and most complex technique of developing portals. There is a strong need for improvements, which would reduce the time and cost of development. Improvements may be made by eliminating problems that occur during development. However, before suggesting solutions, problems must be properly identified and described. Table 6 lists and categorizes all issues associated with the development process presented in section 4.6. Each problem is briefly described from three different perspectives: Code generation, Validation, Visualisation. Code generation problems are associated with the necessity of performing repetitive programming tasks and may be solved by
providing mechanisms for automatic code generation. Validation problems are caused by lack of mechanism, for validating various definitions in terms of domain rules. Visualisation problems are associated with lack of mechanisms for illustrating various SharePoint entities and relationships between them.

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>11. Defining columns</td>
<td>Code generation</td>
<td>Assistance in specifying right attributes, depending on the data type chosen for column.</td>
</tr>
<tr>
<td></td>
<td>Validation</td>
<td>Checking whether values of column attributes are proper.</td>
</tr>
<tr>
<td></td>
<td>Visualisation</td>
<td>Visualisation of grouping columns.</td>
</tr>
<tr>
<td>12. Defining Lookup columns</td>
<td>Code generation</td>
<td>Ability to define lookup columns during design.</td>
</tr>
<tr>
<td></td>
<td>Visualisation</td>
<td>Visualisation of relationships between lists.</td>
</tr>
<tr>
<td>13. Generating content type ID</td>
<td>Code generation</td>
<td>Generation of content type ID.</td>
</tr>
<tr>
<td></td>
<td>Visualisation</td>
<td>Visualisation of inheritance between content types.</td>
</tr>
<tr>
<td>14. Referencing columns in content types</td>
<td>Code generation</td>
<td>Generation of columns references in content type definition.</td>
</tr>
<tr>
<td></td>
<td>Visualisation</td>
<td>Visualisation of inclusion columns and content types.</td>
</tr>
<tr>
<td>15. Adding columns to lists</td>
<td>Code generation</td>
<td>Generation of columns definitions in list definitions.</td>
</tr>
<tr>
<td>16. Adding content types to lists</td>
<td>Code generation</td>
<td>Generation of content types references in list definition.</td>
</tr>
<tr>
<td></td>
<td>Visualisation</td>
<td>Visualisation of inclusion of content types in lists.</td>
</tr>
<tr>
<td></td>
<td>Validation</td>
<td>Ensuring existence view columns in list definition.</td>
</tr>
<tr>
<td></td>
<td>Validation</td>
<td>Ensuring the validity of activation dependencies.</td>
</tr>
<tr>
<td></td>
<td>Visualisation</td>
<td>Visualisation of activation dependencies between features.</td>
</tr>
<tr>
<td>19. Defining site definitions</td>
<td>Code generation</td>
<td>Generation of site definitions with their contents, and site templates.</td>
</tr>
<tr>
<td></td>
<td>Visualisation</td>
<td>Visualisation of contents of site definitions.</td>
</tr>
<tr>
<td>10. Working with SharePoint collections</td>
<td>Code generation</td>
<td>Generation of classes, which assist in working with SharePoint collections.</td>
</tr>
</tbody>
</table>

Table 6. Table of issues in SharePoint development
5 SharePoint DSL

5.1 Introduction

Number of issues and their negative impact on portal development process were reason for searching for tools or techniques, which would support portal development process on SharePoint platform. Unfortunately, such solutions did not exist, and they had to be built. This chapter describes in details process of development and capabilities of Domain-Specific Language proposed in this work – SharePoint DSL. It has been designed as a solution for problems identified in chapter 4, with respect to description of development process and other recommendations from chapter 3. It may be considered as a case study, during which methods and approaches for DSL development were applied in practice. Basing on the outcome of that process, and after its analysis, evaluation of usefulness of proposed language was performed. Sections of this chapter will contain as follows:

- Section 5.2 provides brief description of tool, which was used for DSL development, which includes short specification of its capabilities, advantages and reasons for choosing it for this work.
- Section 5.3 describes process of development of Domain-Specific Language, with respect to one described in section 3.5. Process has been divided into phases (Decision, Analysis, Design, Implementation and Deployment), during which most suitable patterns were applied. This description is also compatible with development process described in [3]. This allows combing theoretical approach to DSL development with real world methodology, by matching appropriate activities and phases. Section 5.4 presents solutions, which have been made in order to solve defined problems. Most of them are realized as diagram elements, based on which various artefacts are generated. Diagram was divided into smaller pieces in order to present solution in clear and readable manner, however those compartments contains all elements relevant for this work. Some of those elements solve multiple problems, while others are useful only in conjunction. For that reason, section was further divided into compartments, in which elements which belong to the same group are described. Additionally, each group of components is summarized by showing their usage on DSL diagram. On figures which presents fragments of Domain Model or Presentation, mappings between those elements were removed for clarity. For same reason, some associations on examples have been also hidden. Additionally, each element is marked by capitalizing its name, while its definition is referenced in text by marking it with monospaced slab serif.
- Section 5.5 summarizes this chapter.

5.2 Description of tool

There are several tools, which supports development of Domain-Specific Languages. Most popular of them are Eclipse Modelling Framework, MetaEdit+ and Microsoft Visual Studio Tools for Domain-Specific Languages (DSL Tools). Those tools enable development of DSLs, which have different functionality – from domain
specific notation only, to extensive frameworks, which supports every aspects of
domain specific development. The tool chosen for the purpose of this work is DSL
Tools.

Figure 11. DSL designer which is running inside Visual Studio

DSL Tools supports development of Domain-Specific Languages, by defining a
modelling language and providing infrastructure necessary to work with it [18]. DSL
Tools exists as a part of SDK for Visual Studio 2005 and 2008, and recently Microsoft
announced its support in upcoming Visual Studio 2010. First phase of development
with DSL Tools is definition of a language, which is done by defining its elements –
class and relationships between them. Each element is then with definition of its visual
representation. Based on those definitions and leveraging underlying engine, visual
designer is generated. Modelling capabilities may be further extended with
mechanisms for generating various kinds of artefacts, such as source code, data,
configuration files, or even other models. Those mechanisms are also supported by
framework shipped with DSL Tools, so there is no need for building code generation
infrastructure from scratch. Conjunction of those two mechanisms - visual designer
(see Figure 11) and code generation - enables specifying entities in domain specific
notation, and then generating artefacts, which represents those entities in general
purpose language. Therefore, DSL Tools fulfil requirements for building External
DSLs.

This tool was chosen mainly because of its integration with Visual Studio IDE.
Developer can benefit from this integration in two ways. First is development of the
language, which is made in familiar environment without necessity of installing
graphical designers or additional tools, except for Visual Studio SDK, which contains
DSL Tools. Integration with Visual Studio is also important, when evaluating
usefulness of developed DSL – DSL Tools allows deployment of languages developed
with it, in form of extension for Visual Studio. This is the main reason for choosing
DSL Tools – Visual Studio is primary IDE for SharePoint development, so integration
of new language with environment, which is familiar to SharePoint developers, is great advantage. Additionally, tasks which are out of scope of DSL may be performed without switching between tools. Last advantage of DSL Tools is the fact that every aspect language developed with it may be further customized. Format of XML, to which diagrams are serialized, may be defined in a way, which enables its migration between other tools. Functionality of visual designer may be enhanced, in order to perform more complex tasks on a diagram. Finally, mechanisms for generating artefacts may be replaced with others, if there is such need.

5.3 Description of process

5.3.1 Decision

The decision in favour of development of Domain-Specific Language was made quite easily. During work with SharePoint technologies in traditional way, programmers encountered many problems, which solving would significantly reduce costs of development. Although there are already tools, which solves some of those problems, their scope is usually limited to one or few issues. There is lack of more complex tool or framework, which would solve more than few issues. Moreover, most of identified problems have no solutions at all. In order to improve development process, some resources were allocated for creating new solution, which is SharePoint Domain-Specific Language dedicated for SharePoint – SharePoint DSL.

The amount of issues, which solution may be achieved by automatic source-code generation is the reason for qualifying the DSL into the Task Automation group of DSLs. This applies to issues in creating columns, referencing them in content types and lists, generating IDs (especially for content types) etc. However, deeper analysis of some problems proves that SharePoint DSL may be qualified as a data structure representation as well, especially when solving such problems as visualisation of content type inheritance, determining column membership in content types and lists, and content type membership in lists. The goal of SharePoint DSL is also to provide tools for architects, which would help them in specifying contents of functionality packages, such as SharePoint features or site definitions. With this kind of support, it is easier to design more re-use oriented components, which may be incorporated into multiple solutions. Such language is close to concept of DSL supporting Software Product Line.

5.3.2 Analysis

The analysis and identification of problems within the domain was conducted with help of SharePoint programmers and architects, who are, in this case, domain experts. Their knowledge and experience, along with technical documentation, and literature is main source of domain knowledge. Additionally, existing software, which supports SharePoint development, was reviewed in order to verify if there already are solutions for some of the problems, and methods of solving those problems were examine as well. Problems and issues, that occurs during development of SharePoint applications were identified long before development of DSL was considered. Nevertheless, those problems were never gathered in one place. For that reason, first step of analysis was to collect information about recurring issues. The next step was to structure the
knowledge about SharePoint development, by describing the process and development
techniques, and identifying artefacts. The last phase of analysis was to describe those
problems and issues, which were going to be solved by DSL.

Analysis was performed without employing formal methods of extracting domain
knowledge. The informal analysis was sufficient, considering the fact, that author of
this work has both knowledge and experience in SharePoint development, as well as
easy access to other domain experts. Despite its informal nature, analysis phase has
the outcome, which consists of definition of domain (sections 4.2 and 4.3) along with
vocabulary of domain-specific terminology (sections 4.4 and 4.5), and description of
problems and issues identified within domain (section 4.6, summarized in Table 6).

5.3.3 Design

The need for visualisation for SharePoint entities and relationships between them
was main reason for choosing graphical notation for the language. Those illustrations
are useful not only for developers, but also for architects, and even for customers.
Working with one diagram instead of set of XML increases productivity of
programmers, while graphical representation of contents of SharePoint deployment
packages facilitates work of architects. Graphical notations are also easier to
understand by layman, so some of the SharePoint entities may be verified by customer
(assuming that he undergoes basic training in using the language).

Traditional languages used for SharePoint development (C#, VB, CAML) lack of
any kind of graphical notation. On the other hand, existing, general purpose modelling
languages, such as UML, are too complex for that purpose, and their adaptation for
defined purposes would be too expensive. For those reasons, SharePoint DSL required
language invention approach, which outcome is External DSL.

Choosing graphical notation and independence from existing languages were
sources for additional requirements for tool, in which DSL was designed. Luckily,
DSL Tools were design especially for graphical language development. Usage of this
tool introduces possibility of applying methods for designing DSLs, which were
described in tool’s documentation and associated literature, such as [3], thus this phase
of the process may be considered semi-formalized. Generally, proposed design
process consists of defining DSL from different perspectives, and then – specifying
mappings between their contents. Practically, DSL Design is performed by defining
DSL on a special diagram, which is divided into swimlanes – Classes and
Relationships, and Diagram Elements – which represents two perspectives from which
DSL may be viewed – entities which it represents, and means used for their
representation. Mappings between elements from those swimlanes define how entities
are visualised on DSL diagrams. All those elements are described in detail in
following sections, while their usage in context of solving particular problems is
presented in section 5.4.

5.3.3.1 Domain Model

First activities focused on gathering terms from within domain, defining their
meaning and relationships between them. It was done by building Domain Model –
graphical representation of domain (Figure 12), which consists of two kinds of entities
- Domain Classes and Domain Relationships. Domain Classes represents concepts
form domain, and are usually presented as shapes. Domain Relationships represent relationships between concepts, and in most cases they are presented as connectors between shapes. However, some relationships can be mapped to physical relationships between shapes, such as containment. DSL Tools enables specification of several kinds of relationships:

- Embedding, which connects container with its contents;
- Reference, which connects source with target;
- Inheritance, which is relationship between derived element and its base element.

There is a third aspect of domain model, but it has no graphical manifestation. This aspect is diagram validity, which represents restrictions imposed on relationships between Domain Classes. On the level of Domain Model, only basic rules may be defined, such as multiplicity of relationships between classes, and their mandatory. Those rules, along with custom rules defined during Implementation phase (see 5.3.4), are verified during diagram validation, which occurs while working with DSL instances.

Figure 12. Fragment of Domain Model used in SharePoint DSL

5.3.3.2 Presentation

The second perspective, from which language is described, is its appearance. Like most graphical notations, DSLs which can be built from DSL Tools consist of building blocks such as various kinds of shapes and connectors. Those elements, along with additional decorators, are laid on two-dimensional drawing surface. This part of DSL design is referenced as design of Presentation [3]. DSL Tools enables defining several kinds of shapes (compartments, images, geometry, and ports) and one kind of connector. It is possible to define several visual aspects of shape, such as colour, size etc. It is also possible to define properties, which determine positioning of the shape within diagram (e.g. swimlane, on which it may be deployed), and properties which
are specific for particular kind of shape (number of compartments, image). It is possible to define appearance of connector as well, for example by choosing shapes, which are located on its ends (arrow, diamond, etc.). Example of definition of Presentation aspect of DSL is presented on Figure 13.

![Diagram Elements](image)

**Figure 13.** Fragment of Presentation definition used in SharePoint DSL

### 5.3.3.3 Mappings

The important thing is that the Domain Model and Presentation are independent, thus they can be designed separately. After each definition is complete, correspondence between Domain Model and Presentation is determined by defining shape Mappings. Mappings are represented by special connectors, which bind elements from Classes and Relationships swimlane, and Diagram Elements swimlane. Those mappings define how domain concepts (Domain Classes and Relationships) will be presented on diagram (shapes and connectors). The ability to separate domain model and presentation enables modifying the graphical representation of the domain concepts without modifying the domain model, and vice versa. Realisation of Mappings is shown on Figure 14.
5.3.4 Implementation

There are essentially no implementation patterns, which completely match graphical DSLs. Most of them describe cases, when existing infrastructure for textual language development is used to create textual domain-specific notation, which will be somehow related to it. The closest and most suitable approach, from those presented in Table 4 is COTS pattern – using existing tools and notations, and applying them to particular domain. In case of SharePoint DSL, tool which would be applied is dedicated for development of DSLs. Furthermore DSL Tools for Visual Studio is suitable not only for design, but also for implementation. Therefore, as it was in case of design phase, methods and approaches taken during implementation will be adapted from the DSL development process described in [3].

DSL Tools, as a tool dedicated for DSL development provides various mechanisms, which supports their implementation. The bottom line is the fact, that there is no need for implementing the language from scratch, because DSL Tools is equipped with engine that does it automatically. Basing on Domain Model, Presentation and Mappings between them, DSL Tools is able to generate very simple language and enables modelling with it. However, some implementation effort is needed, in order to leverage advanced validation or code generation.

5.3.4.1 Validation

The goal of validation is to ensure that all domain rules are abided. Basic domain rules, such as multiplicity and mandatory of domain relationships, may be defined during design phase (5.3.3.1). DSL Tools is equipped with set of libraries, which handles validation of those rules automatically. However, more sophisticated rules must be defined and handled manually. It is done, by developing set of C# classes, which corresponds with domain entities. Those classes are actually only parts of whole definition (partial classes), because core of each class is generated from model by DSL Tools engine. Each class generated from model is code equivalent of Domain Class, along with its properties and representation of relationships with other classes. This allows exploiting inheritance between Domain Classes and other Domain Relationship between them from the level of code. Classes, which host validation
logic, must be marked with `ValidationState` custom attribute along with `Enabled` parameter. The methods that implements validation logic within class must have `ValidationMethod` attribute applied to them. Additionally, set of `ValidationCategory` parameters may be specified, in order to define when validation methods are to be called. Listing 12 presents an example of method, which implements validation logic. The method was developed with respect to recommendations from [3]: validation methods are private and efficient, while error messages are stored in resource file.

SharePoint DSL employs validation mechanism, in order to ensure, that model will be consistent with domain rules. In most cases, rules that are defined on domain model are insufficient. Validation of activation dependencies between features (18) or checking whether attributes specified for fields are proper (11) is done by implementing validation methods.

```csharp
[ValidationState(ValidationState.Enabled)]
public partial class ListTemplate
{
    [ValidationMethod(ValidationCategories.Menu |
                      ValidationCategories.Save)]
    private void ListTemplateValidation(ValidationContext context)
    {
        // verify if list template has at least one view associated
        // and whether there is exactly one view marked as default
        byte viewCount = 0;

        foreach (View view in this.Views)
        {
            if (view.DefaultView == true)
                viewCount ++;
        }

        // there is more than one view marked as default
        if (viewCount > 1)
            context.LogError(Localization.Error117, "Err 117", this);

        // there are no views attached to list template
        if (viewCount < 1)
            context.LogError(Localization.Error118, "Err 118", this);
    }
}
```

Listing 12. Validation method used in SharePoint DSL.

### 5.3.4.2 Artefact generation

Ability to generate code and other artefacts is considered to be most prominent advantage of DSLs. Most of software product artefacts are stored within text files, such as source code, configuration files, scripts for creating and population databases, or documentation. For that reason, DSL diagram transformations should provide functionality of generating those kinds of artefacts relying on information contained within DSL diagram. There are several approaches to model transformation. First is using Extensible Stylesheet Language Transformation (XSLT) on the XML
representation of the diagram. This approach is very flexible, because XSLT is
dedicated for transforming XML documents into any other kind of document.
Additionally, DSL Tools provide mechanisms for customization of the XML format,
in which DSL diagrams will be stored. Second approach is to leverage Domain-
Specific API, which is shipped with DSL Tools. It may be used to manipulate
instances of language directly in memory. The third approach is based on using text
templates, and is the principal technology for generating artefacts in DSL Tools. Most
important assets associated with this method are parameterized text templates, which
are stored within files with .tt extension. Those files contain two kinds of content:
rendered directly to the output file, and contained within special markers, which have
special meaning and are interpreted by transformation engine. Second class of elements
may be further divided into following categories:
- Control blocks: represents parts of template, which are evaluated during template
  transformation. There are three kinds of control blocks:
    o standard control blocks: introduces control statements, such as loops or
      conditions (Listing 13).

Listing 13. Syntax of standard control block

    <# ... #>

- class feature control block: introduces additional elements to template,
  such as methods, properties, fields, or nested classes (Listing 14).

Listing 14. Syntax of class feature control block

    <#+ ... #>

- expression control block: evaluates expression that is contained within it,
  and calls .Net method ToString() on the result (Listing 15).

Listing 15. Syntax of expression control block

    <#= ... #>

- Directives: provides processing instructions to engine that executes text templates
  (Listing 16). There are five types of directives:

Listing 16. Syntax of a directive

    <@@ DirectiveName [ParameterName="Value"] #>

- template: specifies general processing options for text template
  (Listing 17) such as base class for the class that represents template
  (inherits), programming language (Visual Basic or C#) that is used in
  code inside control blocks (language), culture that is used to format
  evaluated values (culture), whether the template should be provided
  with debugging options (debug) and have access to application, that
  hosts text transformations.

Listing 17. Syntax of template directive

    <@@ template inherits="MyNamespace.MyBaseClass" language="C#"
culture="en-US" debug="false" hostspecific="false" #>
- output: specifies type of the file, that will be produced as a result of template transformation, by providing its extension and character encoding (Listing 18).

```
<#@ output extension=".cs" encoding="utf-8" #>
```

Listing 18. Syntax of output directive

- assembly: specifies assembly (name), which contain classes which will be referenced in control code within template (Listing 19). It is equivalent of adding an assembly reference to Visual Studio project.

```
<#@ assembly name="Microsoft.SharePoint.dll" #>
```

Listing 19. Syntax of assembly directive

- import: specifies namespace, in order to use external classes within control code without fully qualifying their names (Listing 20). It is equivalent of using statement in C#.

```
<#@ import namespace="Microsoft.SharePoint " #>
```

Listing 20. Syntax of import directive

- include: specifies file that include part of template (Listing 21). It allows defining one template among multiple files.

```
<#@ include file="FileToInclude.tt" #>
```

Listing 21. Syntax of include directive

- custom directives: provides ability to introduce custom code, also called a directive processor, into template processing.

Artefact generation is very important part of SharePoint DSL, because most of the problems in SharePoint development, defined in Table 6, are associated with lack of mechanisms for code generation. To solve those problems, SharePoint DSL enables generation of number of artefacts. There are two types of artefacts that are: XML files with CAML definitions of SharePoint entities, and files that contain source code in C# language, which represents some of those entities as wrappers for SharePoint object model. Listing 22 presents fragment of text template, which was used for generating XML definition of Content Type. It illustrates usage of several described mechanisms, such as standard end expression control blocks, or import, assembly and output directives.
5.3.5 Deployment

The fact, DSL Tools supports deployment of DSL as an extension to Visual Studio IDE was decisive in choosing the deployment pattern. There are other supporting factors, beside the fact, that this tool was used in previous phases of development process. Integration with Visual Studio, which is IDE for SharePoint development, is the most important advantage of chosen approach – it would allow simultaneous work with all programming artefacts, which are necessary to create SharePoint solution: DSL models, code generated from those models, and hand crafted code. However, creating an extension for Visual Studio and using DSL Tools requires following rules and patterns for creating DSLs, which are described below.

5.3.5.1 Installation package

Designer built with DSL Tools is a Visual Studio Package, which installation requires deploying files in several locations and making some changes in Windows Registry. For that reason, there is a need for building robust installer, which deploy files in right places and make necessary changes in every supported machine configuration. Luckily, building Windows installation packages is supported by DSL Tools, by providing DSL Setup project template. As a result of building this project, an installer package (file with .msi extension) is produced, as well as all other necessary files. A .msi file contains components (such as assemblies), registry settings and references to external files, which are necessary for software installation. DSL developer is able to modify the contents of the installer, as well as customize its

```
<# @ output extension=".xml" #>
<# @ assembly name="SharePointDSL.dll" #>
<# @ import namespace="SharePointDSL.Helpers" #>

<?xml version="1.0" encoding="utf-8" ?>
<Elements
 xmlns=http://schemas.microsoft.com/sharepoint/>
<# foreach(FieldGroup group in fieldGroups) {#>
  <!-- <#= group.Name #> -->
  <![CDATA[
    <![CDATA[
    <![CDATA[

Listing 22. Fragment of text template used in SharePoint DSL
```
layout. After launching .msi file, built-in Windows installer infrastructure takes control over installation process. If multiple .msi files are part of the installation, setup.exe file is needed in order launch them in correct order. SharePoint DSL installation package consist of:

- DSLSharePoint.msi – installation package that contains designer for SharePoint DSL
- DSLToolsRedist.msi – installation package that contains DSL Tools Redistributable – set of binaries, on which all designers created with DSL Tools depends
- setup.exe – installs previous packages in correct order
- Readme.mht – readme file

DSL Tools are also shipped with DSL Tools Installer Definition language. This language is simplified version of standard XML format for defining contents of .msi package, which is called Windows Installer XML (WiX). It focuses only on installing designers and associating components with Visual Studio. Thanks to that only about 30 out of 800 lines of XML has to be hand-written – rest is generated.

5.3.5.2 DSL Tools Framework

Several other components are deployed along with visual designer made for DSL. Those components support modelling and enhance DSL package with other capabilities, such as code generation. Contents of this framework are represented by contents of project, in which DSL is defined. Each DSL project developed with DSL Tools has three-layer architecture [3], which consist of: compiled framework, generated code and hand crafted code (see Figure 15). Compiled framework consists of several assemblies, which provides functionality necessary to work with any kind of Domain-Specific Language:

- Microsoft.VisualStudio.Modeling: manages model elements and links between them in terms of transactions (e.g. undo, redo) and propagation of changes (synchronizing visualisation with internal representation).
- Microsoft.VisualStudio.Modeling.Validation: handles the validation of diagram elements; it creates appropriate errors when validation fails thus providing information about those diagram aspects, which are not coherent with domain rules.

The second layer consists of classes, which are generated from the Domain Model, and classes necessary to integrate created DSL with Visual Studio environment. Most important of them are described below:

- DomainClasses.cs: contains class for each of the Domain Classes defined in the DSL definition.
- DomainRelationships.cs: contains class for each Domain Relationship defined in the DSL definition.
- DomainModel.cs: contains a class, which is representation of the model as a whole, stores information about Domain Classes and Relationships.
- Shapes.cs, Connectors.cs: contains classes, which represents graphical aspect of DSL diagram; overriding methods stored in that class allows customizing behaviour of designer generated for particular DSL.
- Diagram.cs: contains code for updating diagram content when synchronizing with changed model.
- Serializer.cs, SerializationHelper.cs: contains definition of the methods of saving models as XML files.
- Commands.ctc, GeneratedCommand.h, CommandSet.cs, Package.cs: stores definitions and manage commands that will be available in Visual Studio after installing DSL.
- EditorFactory.cs, DocData.cs, DocView.cs, ModelExplorer.cs: manages Visual Studio behaviour, when DSL diagram is opened, changed or saved.

The third layer contains mainly logic for validating model and its contents, which was created during Implementation phase. Validation logic is specific for particular DSL, thus it may be stored in many ways. Additionally, some extensions of the designer framework may be implemented, in order to provide more ways of working with diagrams.

Figure 15. Architecture of DSL Tools, from [3]

5.4 Mechanisms for solving problems

Solutions for problems, which were categorized as code generation issues, and are associated with generating CAML definitions of various SharePoint entities were
based on CAML schema. Fixed parts of CAML definitions are stored in text templates, while variable parts, such as values of attributes or child nodes are filled during model transformation. In most cases, values are acquired from shape properties, and child nodes are represented by shape contents or connections with other shapes.

5.4.1 Swimlanes

On the highest level, diagram is divided into compartments called swimlanes. SharePoint DSL consists of four swimlanes: Fields (FieldsSwimlane), Content Types (ContentTypesSwimlane), Lists (ListsSwimlane) and Deployment (DeploymentSwimlane). Their main purpose is to provide mechanisms for both visual and logical partition of the diagram into smaller pieces, which are easier to manage, and illustrate different functional capabilities in more convenient way. Within each swimlane, it is allowed to store only those elements of the diagram, which are logically associated with it. For example, Fields swimlane may contain only those diagram elements, which serves as columns definitions or assets associate with columns. On the other hand, in some cases elements, which are stored within different swimlanes, may be associated with each other. The example of such relationship may be association of column definition with feature, which deploys this column. Those two types of SharePoint entities are stored within separate swimlanes (see 5.4.2 and 5.4.5), yet there is need for some way of showing this kind of relationships. In order to meet this need, there is a group of connectors, which may bind elements from different swimlanes. Figure 16 presents part of domain model, which defines swimlanes. Swimlanes themselves do not solve any of the problems defined in section 4.6 – their purpose is to make the language easier to use, by providing way of partitioning diagrams and grouping elements.

5.4.2 Columns

There is only one type of shape that may be part of Columns swimlane – Fields Group (FieldsGroup). This shape is a representation of SharePoint group for columns, which purpose was stated in section 4.5.1. Fields Group shape serves also as storage for Fields on a diagram. Figure 17 presents part of Domain Model, which
defines contents of Fields swimlane. Within each Fields Group, multiple fields - both pre-defined (WSSField) and created from scratch (SPField), may be defined. SPField Domain Class serves as base class, for Fields of particular data type (such as Text, Boolean, Number, or Lookup).

Figure 17. Domain Model - Fields swimlane

The column definition is performed by adding new Field to the Group, and assigning values to all necessary properties. Set of properties is variable – it depends on the data type chosen for Field, which is determined during defining Field within Group. In case of defining Field, which data type is Lookup, it is possible to choose List and Field which it reference. If the values for those properties are not right (e.g. out of scope), an error will occur during diagram validation. During the transformation of the valid model, basing on the appropriate text templates, XML definitions of columns will be generated. When some of those columns are Lookups, their definitions will contain special tags with temporary values. During columns’ deployment, temporary values will be replaced with actual values, by invoking custom code (shipped along with the language), which will acquire those values from SharePoint environment. Described mechanisms provide solutions for all aspects of problems associated with defining columns for both defining columns (I1) and defining lookup columns (I2).
Figure 18. SharePoint DSL example - Fields swimlane

Figure 18 presents an example of Fields Group (Employee Columns), and some columns which are defined within this group (Name, Last Name, Gender etc.). Properties of columns, especially those dependent on chosen data type (such as Format or Choices for Choice data type), may be set in Properties window.

5.4.3 Content Types

All contents of Content Types swimlane and relationships between them are presented on Figure 19. The most important diagram element that may be defined within Content Types swimlane is shape which represent content type (AContentType). There are two classes of Content Types – predefined (WSSContentType) and custom (SPContentType). Custom Content Types are defined by adding appropriate shape to diagram and defining its properties such as name, description and group. As it was stated in section 4.5.3, every custom content type must inherit from other content type. Content type inheritance (SPContentTypeInheritsAContentType) is represented by special connector (ContentTypeInheritConnector), which binds child Content Type with its parent. SharePoint DSL enables inheriting form content types, which are defined within the diagram, and those shipped with WSS. When model is transformed, IDs for content types are generated automatically. Those mechanisms are solutions for both code generation and visualisation aspects of problem I3.
The second important part of defining content types is to specify which columns are parts of it. It can be done by simply adding references to the columns inside content type shape, and choosing particular column by picking its name from drop-down list. Relationship between Content Types and Fields are defined by AContentTypeHasFieldRef Domain Relationship. It is also possible to change Fields order within Content Type shape, which reflects the order of columns in SharePoint UI. This behaviour is solution for visualisation aspect of problem I4. The model transformation is the mechanism, which solves code generation issues associated with I4. During model transformation, appropriate field references will be automatically generated inside content types. Additionally, columns which are inherited from parent content types are automatically added to content type definitions, so there is no need for do it manually neither on the model, nor in XML definitions.
Figure 20. SharePoint DSL example - Content Types swimlane

Example on Figure 20 presents simple inheritance hierarchy, where Item (ContentType shipped with WSS) is parent for custom ContentType - Employee, which has one child – Temporary Employee. Each of those ContentTypes have set of Field references, which are added by creating new reference inside ContentType shape, and choosing one of defined Fields from drop down in Properties window. ContentType contains only new field references – there is no need for adding references, which are defined in its parent.

5.4.4 Lists

List swimlane may contain several different elements of diagram. Most important is List Template shape, defined by ListTemplate Domain Class, which representation of actual definition of SharePoint list. Like in the previous cases, there is number of properties associated with this shape, which are equivalents of list properties, such as its base type, name (internal and visible via UI), configuration of versioning, content type management and many more. List Template does not solve any of problems by itself, because specifying list attributes in list schema is equally complex. However, List Template is the important endpoint for connectors, which binds it with diagram elements such as ContentTypes, Views, and List Instances. Connector between List Template and ContentType (ListTemplateReferenceAContentType) is representation of inclusion of content type within list definition. After transformation of model, the XML list definition will contain references to all ContentTypes connected to the List Template, along with copies of definitions of all Fields, which are referenced in those ContentTypes (ListTemplateReferencesAField). With this one connector (relationship between list template and columns has no visualisation), both defining columns (I5) and referencing content type (I6) issues are solved, in code generation and visualisation contexts. All mentioned classes and relationships are shown on Figure 21.
Figure 21. Domain Model - Lists swimlane - List Templates

Second connector (ViewReferencesListTemplate) binds List Template with View shape (View). Views are representation of definitions of SharePoint views, which are normally part of list schema. However, in order to make those definitions more manageable, they are separate shapes. Beside specifying attributes of the View, such as its name or type, the properties of shape includes query for the view, limit of view rows. The most important function of the View shape is to store the set of view columns (ViewHasFieldRef). View columns are added to view as references of Fields, which are added to List Template via Content Type connection (ListTemplateReferenceAContentTypes). This ensures the existence of column within list definition, and along with generating view definitions within appropriate list schemas during model transformation, is fulfilment of requirements stated in I7.

Last important shape from List swimlane is List Instance, which is representation of the instance of the Share Point list. It must be connected to the List Template on which the instance is based (ListInstanceReferenceListTemplate). Shape’s properties contain such attributes as instance name, description and URL. Its existence is motivated by the fact, that list instances are often listed as elements of features, thus they are needed for solving issue I8, which will be discussed in details in section 5.4.5. Fragment of Domain Model, which defines Views, List Templates and relationships related to them is shown on Figure 22.
Example of diagram elements related to SharePoint lists is shown on Figure 23. Central shape is Employees List Template, which reference Employee Content Type. There are two Views associated with List Template – All Employees and Senior Employees. Both views have the same set of View Columns, which are chosen from columns defined within Content Types referenced in List Template. The difference between Views is query, which filters elements that will be accessible via particular View. There is also the Instance of List Template, which is also named Employees.
Properties of each of those shapes (such as Base Type for List Template) may be set in properties window.

5.4.5 Deployment

The most important shape that may be stored within Deployment swimlane (Figure 24) is Feature. Properties of this shape allows setting values for many attributes specific to SharePoint feature, such as its scope, sequence in which it should be activated, receiver assembly and class. However, like in case of List Template shape (5.4.4) its main aim is not enabling specifying those properties in more convenient way, but to enable binding it with other shapes, such as Columns Groups, Content Types, List Templates and List Definitions. All elements, which may be listed in feature manifest, are represented by children of FeatureContents domain class, and relationship between feature and its contents - by FeatureReferencesFeatureContents domain relationship. After model transformation, each element connected to Feature will be listed in manifest of elements, which are to be deployed along with it. Additionally, when list definition is to be deployed by the feature, the complex hierarchy of directories, mentioned in section 4.5.4, is created automatically. This functionality is solution for code generation aspect of issue I8.
content types, and lists, can be reused as well. Features and Feature Activations shapes may be connected with each other by Activation Dependency connectors \((\text{FeatureDependsOnAFeatures})\). Those connectors represent links between feature and all features on which it depends. Additionally, there are mechanisms for verifying, if dependencies are valid, considering features’ scope and visibility. Those mechanisms provide solution for visualisation and validation aspects of problem I8.

Both Features and Feature Activations may be connected with Site shape (Figure 25) that represents site definition \((\text{SiteContainsAFeatures})\). This relationship illustrates inclusion of features within configuration associated with site definition, which is solution for visualisation aspect of problem I9. SharePoint DSL enables creating only one configuration per site definition, because it is sufficient for most business cases. During model transformation, directory hierarchy along with \text{onet.xml} and \text{webtemp.*.xml} files is created within project. It is important, that reference to site configuration in \text{webtemp.*.xml} is created automatically. Additionally, basing on relationships and activation sequence, which is defined for each feature, features are referenced in appropriate sections of site configuration. Those mechanisms solve code generation issues identified in I9.

![Classes and Relationships](#)

**Figure 25. Domain Model - Deployment swimlane – Sites**

Figure 26 presents contents of deployment swimlane. Central element is Company Site definition, which incorporates three Features – Content Types, List Templates and List Instances. List Instances feature contains employee list, which is illustrated by dark blue connector between feature and list. Content Types Feature depends on activation of two other Features (light blue connector) – Fields and MOSS Fields. Second feature appears as Feature Activation (red frame). Each of Feature shapes exposes some of its properties, such as sequence (above shape) and scope (below shape).
5.4.6 Wrappers

Wrappers are assets, which are second kind of artefacts generated from code. Their purpose is to provide more convenient and reliant way to access SharePoint entities via object model. Particularly, it provides values of attributes, which identifies concrete objects on SharePoint collections. Wrappers are essentially C# classes, which exposes set of properties, which represents attributes of SharePoint entities. Values of those attributes are defined on diagram or during XML definitions generation, so it is possible to generate additional artefacts, such as classes which contain those values – wrappers. Before introducing DSL, many SharePoint developers had to develop similar classes manually, by populating its properties by copying appropriate values from XML definitions. Wrappers are mechanism, which automates this activity, and reduce costs of development. They also reduce costs of modifications – once properties of SharePoint entities are changed, wrappers, with new values of properties, are generated again. Moreover, it is possible to pass SPItem object to constructor of such wrapper – then it will be possible to access values of fields of this item in more convenient way, closer to object oriented programming paradigms. Listing 23 presents fragment of text template definition for wrapper for content type. Class generated from this wrapper contains properties for content type name (ContentTypeName), content type id (ContentTypeId), and set of properties, which return objects (FieldInfo) representing columns, which are referenced in content type.
Listing 24 presents example of working with SharePoint object model, with assist from wrappers. This sample corresponds with presented on Listing 11. Wrappers

Listing 23. Fragment of text template used for Content Type Wrapper generation

Listing 24 presents example of working with SharePoint object model, with assist from wrappers. This sample corresponds with presented on Listing 11. Wrappers
enable very simple access to values of properties, which distinguishes SharePoint entities in collections — for example in order to acquire Id of content type, static ContentTypeId property of class which represents this content type should be invoked. This class may be also utilized to acquire identifiers of columns, are referenced in content type, but easier way of acquiring values of particular fields of item, is passing it into wrapper constructor, and access them via its properties.

```csharp
class WSSCollectionsDSLSample
{
    static void Main()
    {
        string siteUrl = "http://example.com";
        SPSite site = new SPSite(siteUrl);
        SPWeb web = site.RootWeb;

        // acquiring list, which name is Employees
        SPList employeesList = web.Lists["Employees"];

        // Id of Temporary Employee content type
        SPContentTypeId temporaryEmployeeCTId = TemporaryEmployee.ContentTypeID;
        // Id of Contract End Date column
        Guid contractEndDateId = TemporaryEmployee.ContractEndDate.Id;
        // Id of Gender column
        Guid genderId = TemporaryEmployee.Gender.Id;
        // Id of Last Name column
        string lastNameDN = TemporaryEmployee.LastName.Id;

        // checking condition for every item on list
        foreach (SPListItem item in employeesList.Items)
        {
            if (item.ContentType.Id == temporaryEmployeeCTId)
            {
                // using wrapper as a object wrapping SharePoint item
                TemporaryEmployee tempEmployee = new TemporaryEmployee(item);
                string message = String.Format("Employee {0} ends (1) contract on {2}",
                    tempEmployee.LastNameValue, ,
                    tempEmployee.GenderValue == "Male" ? "his" : "her",
                    tempEmployee.ContractEndDateValue);
                Console.WriteLine(message);
            }
        }
    }
}
```

Listing 24. Example of using wrappers while working with SharePoint collections
5.5 Summary

SharePoint DSL is Domain-Specific Language, which supports portal development on SharePoint platform, and as such it provides solution for many problems associated with it. Language proves to be successful because of its usefulness in SharePoint development, which was achieved thanks to taking systemized approach to its development. Process was conducted according to the definitions and recommendations found in literature. All those guidelines are based on successfully implemented and deployed DSLs, as well as on standards and patterns from other areas of software engineering, such as development of programming tools or General-Purpose Languages. Additionally, some suggestions of supplements for development process were proposed for Deployment phase, which lack of literature support.

The tool, which was used during that process – DSL Tools for Visual Studio – proved to be a great asset in DSL development. As a tool dedicated for development of DSL, it supports the process in many phases from design to deployment. Its most prominent advantage was the fact, that developed DSL could be deployed and used in conjunction with the development environment, which is used in SharePoint development.
6 Quality of software products

6.1 Introduction

This chapter provides assessment of influence which has SharePoint DSL on SharePoint software products. This evaluation is most important part of this work, because it proves that effort and resources allocated into development of DSL have not been wasted, and proposed language is valuable asset, which supports and enhances software development process. In order to provide reliable assessment, formal method of evaluating software product quality was applied. Contents of this chapter are as follow:

- Section 6.2 provides definitions of terms such as quality, quality model or software quality. Proper understanding of those terms is necessary for describing evaluation method and applying it for this work.
- Section 6.3 describes design of 3 crucial parts of this section – criteria for choosing elements of quality model, selection and description of methods for evaluation, and detailed description of sources of data to evaluate.
- Section 6.4 contains detailed description of quality model designed to evaluate SharePoint software products. It contains justification for scope of quality model, characterize sources of data, which will be used during quality evaluation, and illustrates which parts of base quality model were chosen to construct quality model for SharePoint software products.
- Section 6.5 presents detailed evaluation divided into 6 subsections – first two describes each characteristic and subcharacteristic in detail. This presentation consists of their standard definitions and expansions of those definitions, which were proposed in order to fit pieces of quality model into domain. It also include justification for choosing characteristics and subcharacteristic. Third subsection presents definitions of measures for evaluating quality, while following - results of actual assessment. Last subsection is commentary for evaluation outcome, along with evaluation of overall quality.
- Section 6.6 focus on finding threats to validity of the research. It considers two kinds of validity threats, each of which can affect validity in different area. Section include both definitions of validity threat kinds and validity threats themselves, as well as their location within this work and potential influence on measurement of quality model subcharacteristics.
- Section 6.7 summarizes this chapter.

6.2 Definitions

6.2.1 Quality

According to [25] quality is “degree or grade of excellence”. Quality states how close are some attributes or properties of an entity to desired value. This definition may be adapted to software engineering, where software quality describes how well software is designed, how efficient it performs, to what extent it fulfils the requirements etc. Software quality is very difficult to determine, because it can be viewed from many different perspectives. Additionally, there are many ways of
measuring aspects of software products, and those measures can be made and compared in different contexts.

In order to structure and systemize the various aspects of software quality, International Organization for Standardization (ISO) has developed series of standards for the evaluation of software quality - ISO/IEC 9126, which is to be superseded by ISO/IEC 25010. In those standards, quality is defined as “the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs” [13], whereas software quality is “degree to which the software product satisfies stated and implied needs when used under specified conditions” [14].

6.2.2 Quality model

Quality model is “defined set of characteristics, and relationships between them, which provides a framework for specifying quality requirements and evaluating quality” [13]. The main goal of this model is to provide tool, which may be used for objective evaluation of software quality. Software quality characteristics are categories of software quality attributes, which of affect software quality. Quality characteristics are further divided into sub-characteristics in order make the model more manageable. Software quality attributes, which are grouped within characteristics and sub-characteristics are properties of an entity that can be evaluated quantitatively or quantitatively. Structure of quality model is shown on Figure 27. Quality model for software products, which was defined in ISE/EIC 9126 standard, is shown on Figure 28.

Quality models are primarily starting point for defining specific quality model, which is tailored for particular case. Specifying and measuring quality attributes for all characteristics and subcharacteristics, is often not needed, or even not possible. Relevant characteristics should be chosen basing on the product and application domain. When software is to be used in systems, which may influence users’ health, including in model characteristics associate with security and reliability should be considered in the first place. Properly defined quality models are valuable assets in specifying requirements, defining measures and performing evaluations.

![Figure 27. Structure of quality model, based on [14]](image-url)
6.2.3 Software quality models

Software may be evaluated considering various aspects, such as stakeholders, for whom the evaluation is made (e.g. developers, testers, end users) or type of measure (e.g. inspection of static attributes, test of dynamic attributes). For that reason, ISO proposed three quality models: Software Product Quality Model, which consist of internal and external measures of Software Quality, Data Quality Model and Quality in Use Model [14]. Using those models, software quality may be evaluated from various perspectives (Figure 29).
Internal quality (internal measure of software quality) is defined as “measure of the degree to which a set of static attributes of a software product satisfy stated and implied needs when the software product is used under specified conditions.” [14] Requirements for internal quality are used to specify properties of intermediate software products (specification, source code) and properties of deliverable, non-executable products such as documentation or manuals. Internal quality measurement is performed by inspection of static attributes of software product, when it is not executed. The outcome of measurement is useful primarily for developers and maintainers.

External quality (external measure of software quality) is defined as “measure of the degree to which a software product enables the behaviour of a system to satisfy stated and implied needs when the system including the software is used under specified conditions.” [14] External quality measured when software is executed, usually while testing in simulated environment using simulated data. Therefore, stakeholders who are most interested in this kind of quality are testers and operators. Requirements for external quality are targets for technical verification and validation of software product.

Quality in use is defined as “degree to which a product used by specific users meets their needs to achieve specific goals with effectiveness, efficiency, flexibility, safety and satisfaction in specific contexts of use.” [14] It evaluates quality of software, which is running in simulated or real operating environment, along with hardware, and operating environment itself. Stakeholders, who are interested in this
type of quality, are acquirers, owners and end-users. Requirements for quality in use are derived from stakeholders needs.

Relationship between quality measures, especially between internal, external and in use quality may be used to predict other type of quality. Evaluation of internal quality may be used to predict external quality, and evaluation of external quality – to predict quality in use.

6.3 Design

6.3.1 Quality model design

First step in designing quality model was choosing the right perspective. Main aim of the model was to evaluate software products, which are developed for SharePoint. For that reason, model was designed to compare SharePoint software products developed with and without SharePoint DSL in terms of their internal quality. The reason for that is the fact that SharePoint DSL has little or no influence on dynamic aspects of system – it was design mainly for modelling data structures and deployment packages. Moreover, differences between software products developed with and without DSL language are visible for architects and developers. Scope for proposed quality model is marked on Figure 30.

![Figure 30. Scope for quality model](image)

Second step was choosing the subset of quality characteristics and subcharacteristics, which are essential part of the model. As in the previous case, characteristics and subcharacteristics had to be chosen considering areas, in which SharePoint DSL potentially improved software products. For that reason, the key of choosing characteristics was their association with static aspects of software products, such as stability, operability or maintainability.
6.3.2 Evaluation design

In the second stage of study design, methods for evaluating each subcharacteristic had to be designed. There were several factors, which had to be considered. First and the most obvious was the nature of the subcharacteristics – some of them, such as attractiveness or ease of use are strictly subjective, so the only way of evaluating them was to design and conduct survey. Survey was designed as a series of rating questions, which allowed capturing varying degrees of emotion about subject. The ranking questions were to be answered by assigning numbers (from 1 to 10), which represents level of different notions. Low numbers were associated with negative feelings, while high numbers – with positive. In order to ensure surveys quality in terms of understandability and clarity, a pilot survey was conducted, during which some remarks and recommendations were collected and applied. The detailed description of surveys were included in section 6.3.3.2 and in Appendix B.

Second factor which had to be taken into consideration was to maximize usage of already collected data. Some of the data useful in measuring subcharacteristics was already collected, and in order to evaluate them, appropriate information had to be extracted and processed. Those information were extracted from storages, which held data associated with various projects – those storages are wider described in section 6.3.3.1.

Last part of the evaluation design was to prepare method for measuring those subcharacteristics, which were not appropriate for surveys or data extraction. Considering the chosen perspective of quality (internal quality) which is strongly tied with source code, the analysis of various software product artifacts seemed to be proper choice. As in the previous case, already conducted projects proved to be valuable source of artifacts to analyze (see section 6.3.3.1). On the other hand, some of the dedicated artifacts had to be prepared from scratch – mostly because of lack similar examples in real life projects (see section 6.3.3.3).

6.3.3 Sources of data

In order to evaluate each characteristic form quality model, data associated with those characteristics must be collected and analysed. Sources of data where chosen in manner, which ensures their validity and representativeness of evaluation. For the purpose of this work, three types of sources were chosen, and each of them is briefly characterized in following sections.

6.3.3.1 Projects

The main source of data for assessing quality of SharePoint software products are projects, which were developed in SharePoint technologies. All those projects were developed by IT-Dev - software development company from Wroclaw, which specialize SharePoint development. Six projects were selected to analyse. Three of those projects were developed, when company was using standard development techniques – hand-written CAML definitions – without support for code generation. The second half was conducted recently, after introducing SharePoint DSL into development process. All products made during those projects are very much alike – they have similar functionality, complexity, and were aimed in the same market segment. Those products were intranet portals made for medium-sized companies,
with elements of document and information management. Projects consisted of 2 or 3 site definitions, which contains from 25 to 30 content types distributed among 3 lists. Apart from features necessary to deploy list and content types within site definition, projects contains features, which installs additional functionality, such as custom business logic, which is triggered by adding or making changes in elements on lists, custom navigation structures, custom controls, which increase product usability and provide user interface for some functionalities. Due to confidentiality agreements signed with customers, specific details of the functionality of those projects could not be presented in this work. It is worth mentioning, that all projects were also managed using the same method – applying parts of Microsoft Solutions Framework (MSF).

The fact that all projects consist of set of artefacts of same types allows their comparison from different perspectives. One of them presents project as a set of tasks, which must be completed in order to finish it. Those tasks were specified, tracked, managed and analysed with combination of three tools – Microsoft Office Project 2007 (MSP), Microsoft Office Project Server 2007 (MSPS), and Microsoft Project Web Access (PWA), which are parts of Microsoft Enterprise Project Management (EPM) solution. During project planning, tasks, which usually are associated with particular SharePoint components, were identified and scheduled using MSP. Appropriate data was deployed on MSPS, from where it might be accessed via both MSP and PWA (see Figure 31). With each completed tasks, its metadata was specified. Metadata includes information such as:

- Dates of task beginning and end
- Task completion time
- Task priority
- Cost of task realization
- Resources that were assigned to task
- Summary, which briefly describes aim of the task
- Comments, which describes issues and problems during task realization

Metadata collected from multiple tasks and various projects is valuable asset – its analysis helps to improve planning, estimating costs and time for future projects, and assist in identifying bottlenecks for projects.
Second perspective, from which project may be viewed is set of artefacts developed during project realization. Those artefacts are, in majority, development artefacts such as source code, configuration files, scripts, and after introducing SharePoint DSL – diagrams. However, they also include various documents such as analysis, requirements specification etc. In contrast to task perspective, this one is more concrete – it defines project as a set of physical entities, which are results of realizing particular tasks. It illustrates the method, which was used to accomplish task, while tasks from previous perspective focus on metadata. Artefacts may be analysed, in order to find commonalities among various projects, define set of good practices and identify common problems.

### 6.3.3.2 Survey

Survey is secondary source of data. It was conducted among group of 10 developers – employees of IT-Dev. Most of them are Microsoft Certified Specialists (MCTS) in category “Microsoft Windows SharePoint Services 3.0 – Application Development”, and they have average 2 years of experience in SharePoint development, thus they are reliable source of information. However, developers who are novice in SharePoint development, also provide valuable information, and introduces other, fresh point of view on SharePoint development. Survey provided information complementary to those from project analysis, and enabled evaluation of subjective quality subcharacteristics, such as attractiveness or ease of use. Whole
survey is attached to this work in section, while they results are presented in appropriate charts.

6.3.3 Test

Third source of data was a test conducted among SharePoint developer. Subjects of test were also members of the group, among which survey mentioned in previous paragraph, was performed. Test was chosen as a method for acquiring data for those subcharacteristics, which should be evaluated quantitatively, yet data from projects was insufficient for doing so. The basis for test was set of SharePoint development artefacts, such as source code, XML definitions and DSL models. They were divided into two groups – those developed with SharePoint DSL, including DSL model, and those developed with standard techniques. They were also grouped using Visual Studio projects mechanism. Structure of development artefacts used in test imitates real SharePoint projects, which also consist of various artefacts grouped in Visual Studio solution.

During test, developers were asked several questions. In order to answer them, they had to analyse given set of artefacts. One of those questions was to list all SharePoint data structures, which are installed by given feature. In order to do that, subject of test had to look through appropriate artefacts, find feature and identify all of its contents. In projects developed in traditional way, multiple XML files had to be examined, while in projects based on DSL, analysis of the model was sufficient. Other questions are presented in sections, which presents evaluation of particular quality subcharacteristics. This approach allows test to be as close to the real projects as it is possible, thus provide reliable data to analyse.

6.4 Quality model for SharePoint software products

6.4.1 Quality characteristics

This section presents quality model developed for the purpose of evaluation of SharePoint software products quality. The model is based on the proposal for new software product quality model from upcoming standard ISO/IEC 25010, which will replace ISO/IEC 9126. Figure 32 and Figure 33 presents first part SharePoint software product quality model, as a subset of characteristics (marked grey) and subcharacteristics (marked bold) form software quality model defined in [14]. Motivation for choosing each characteristic and subcharacteristic will be presented in section 6.5. Division of quality model for software product was made for clarity and readability.
Figure 32. Software product quality model, part 1, based on [14]

Software product quality

- **Functional suitability**
  - Functional appropriateness
    - **Accuracy**
    - Compliance

- **Reliability**
  - Maturity
    - Availability
    - Fault tolerance
    - Recoverability
    - Compliance

- **Performance efficiency**
  - Time behaviour
    - Resource utilization
    - Compliance

- **Operability**
  - Appropriateness recognisability
  - Learnability
  - Ease of use
  - Attractiveness
    - Technical accessibility
    - Compliance

Figure 33. Software product quality model, part 2, based on [14]

Software product quality

- **Security**
  - Confidentiality
  - Integrity
  - Non-repudiation
  - Accountability
  - Authenticity
  - Compliance

- **Compatibility**
  - Co-existence
    - Interoperability
    - Compliance

- **Maintainability**
  - Modularity
    - Reusability
    - Analysability
    - Changeability
      - Modification
      - Stability
    - Testability
    - Compliance

- **Portability**
  - Adaptability
  - Installability
  - Replaceability
  - Compliance
6.5 Evaluation of quality of SharePoint Software products

6.5.1 Characteristics

6.5.1.1 Functional suitability

Functional suitability is “the degree to which the product provides functions that meet stated and implied needs when the product is used under specified conditions.” [14] It is set of attributes that bear on the existence of a set of functions and their properties. The main aim of those functions is to satisfy stated needs and fulfil requirements. Like in all areas of software development, the main aim of development process of SharePoint-based portals is to provide product which complies with requirements. One of the expected improvements, made by introducing DSL into development process, was increase in compatibility between SharePoint software products and their specification, and that is why this characteristic was chosen for evaluation.

6.5.1.2 Operability

Operability is “the degree to which the product has attributes that enable it to be understood, learned, used and attractive to the user, when used under specified conditions.” [14] It is a set of attributes that bear on the effort needed for use, and on assessment of such use, by a set of users. It may be also referred as convenience or practicality of use. As it was indicated before, SharePoint software products are complex applications, which development requires extensive knowledge and experience, thus SharePoint development is considered to be difficult. One of the advantages of using DSLs is simplification of development process, in this case by enabling working with graphical diagrams instead of XML files. Several subcharacteristics of operability were chosen in order to verify this hypothesis.

6.5.1.3 Maintainability

Maintainability is “the degree to which the product can be modified.” [14] Those modifications include corrections, adaptation and improvements of software product. There are many reasons for modifications, such as changes in environment or in requirements. Software product is considered to be maintainable, when it is not complex and is well documented. High degree of maintainability makes future maintenance easier, and allows coping with a changed environment. One of the prominent advantages of SharePoint DSL is decreasing amount of work needed for both develop and maintain SharePoint software products. Evaluation of maintainability subcharacteristics was med in order to confirm, that software products developed with SharePoint DSL are easier to maintain, than those developed using standard techniques.
6.5.2 Metrics

6.5.2.1 Accuracy

Accuracy is “the degree of correctness or freedom from error.” [14] It is the degree of closeness of functionality provided by software product, to requirements stated in software specification. The reason for measuring this aspect of software quality is to verify if proposed DSL introduced improvements, which are associated with involving customers into development process. Accuracy as a degree of correctness to customers’ needs and requirements is suitable subcharacteristic for such task.

6.5.2.2 Appropriateness recognisability

Appropriateness recognisability is “the degree to which the product provides information that enables users to recognise whether the software is appropriate for their needs.” [14] It is also known as clarity of purpose or understandability. Understandability is subjective, because different classes of users may have more or less problems with understanding, for example IDE, which is to be used by programmers is required to be understandable particularly by them. The reason for measuring this aspect of software quality is to verify if proposed DSL assist developers, especially those who have little experience and domain knowledge. Learning XML schema for CAML language, which is used in defining SharePoint data structures and deployment packages, is long-term task. SharePoint DSL was designed to eliminate such need by providing more readable notation for defining those entities. Increase in recognisability proves usefulness of the language in this area.

6.5.2.3 Learnability

Learnability is “the degree to which the product enables users to learn its application.” [14] It is capability of a product or tool to enable the user to learn how to use it, by performing basic tasks. In case of SharePoint software products, learnability may be defined as a capability of software product to enable programmers to learn what the purpose of its components is. Like in appropriateness recognisability, learnability is useful mainly for developers, who have not worked in particular projects, or who are novice in SharePoint development. Those two groups require clear description of purpose of each component, which may be provided by source code analysis or documentation. However, SharePoint DSL enables storing both documentation and implementation in one place – as a diagram. This reduces time of learning the purpose of each component, and makes project more manageable.

6.5.2.4 Attractiveness

Attractiveness is “the degree to which the product is attractive to the user.” [14] It is the capability of product to attract its user, make him feel comfortable with using it, increase the pleasure and satisfaction of working with it. Attractiveness is difficult to measure, because it is strictly subjective – it depends on sense of attractiveness, which is unique to every human. In case of SharePoint-based software products, the attractiveness of development artefacts will be evaluated. Attractiveness has huge,
psychological impact on developers’ performance – their productivity seems to be higher when they are working with software, which is considered to be attractive. When they are working with attractive component, they try to maintain this state. Considering the fact, that attractiveness is linked to understandability, clarity of purpose and other operability subcharacteristics, their high level is maintained as well. SharePoint DSL was designed in order to provide notation more attractive than XML files, so attractiveness is evaluated in order to verify if this goal was achieved.

6.5.2.5 Ease of use

Ease of use “the degree to which users find the product easy to operate and control.” [14] It is second subcharacteristic, which is purely subjective. For the purpose of this work, ease of use will be understand as the degree to which developers find intermediate SharePoint software products easy to work with. Psychological attitude towards software may influence developers in many areas, and as such should be evaluated. It applies especially to those situations, when software is complex, and this complexity may be source of negative attitude to work, and so reducing productivity. High level of ease of use may be also useful in reducing time needed, for training new SharePoint developers. In such situations, DSL diagrams, which are easier to work with than XML files, may be great training material and source of knowledge about SharePoint entities. Simplifying SharePoint development was one of the aims for proposed DSL, thus this characteristic has been chosen to evaluation in order to verify, if this goal was achieved.

6.5.2.6 Reusability

Reusability is “the degree to which an asset can be used in more than one software system, or in building other assets.” [14] It may be also defined as the likelihood a part of the software product that can be used again. Reusability is important for reducing time and costs for software development – once developed peace of functionality may be used in many places. It is also useful managing multiple projects, because changes in component are propagated in every project in which it is used. SharePoint provides many mechanisms, which supports reusability, most notable features and site definitions (described in section 4.4.3 and 4.4.4). However, without additional support, capabilities of those mechanisms may not be fully utilized. SharePoint DSL was designed to support taking advantage of environment’s mechanisms for reusability, so evaluation of this characteristic is necessary for proving its usefulness in this area. Measure of reusability in context of SharePoint software products will be based on those mechanisms.

6.5.2.7 Changeability

Changeability is “the degree to which the product enables a specified modification to be implemented. The ease with which a software product can be modified.” [14] Changeability is important aspect of software development, because situation when piece of functionality remains unchanged during whole product lifecycle is very rare. Moreover, changes are introduced during various phase of development process. Modifications made later are more expensive and difficult, even if they concern smaller pieces of functionality. In order to divert resources into more creative tasks,
introducing changes should be as easy as it is possible. Influence of DSL on this aspect of software quality was determined by evaluating and comparing changeability of software products developed with and without it.

6.5.2.8 Modification stability

Modification stability is “the degree to which the product can avoid unexpected effects from modifications of the software.” [14] In other words, it is the capability of the software product to remain stable after some changes are introduced. The fact, that changes are made almost constantly, and that there are relationships between components which are changed, requires verification if changes in one component would not negatively affect the other. The more components are within project, and more complex relationships are defined between them, the more difficult is verifying, if there will be some side effects, whereas probability of destabilizing other components is higher. Modification stability is very important in complex environments, including SharePoint. Additionally, there are many SharePoint components, which are vulnerable for changes in others – it is caused by relationships between them. Graphical notation in conjunction with code generation increased software products stability, which was proved during evaluation of this quality subcharacteristic.

6.5.2.9 Overall Quality

Overall quality is comprehensive measure of quality factors defined by each characteristic and subcharacteristic, and is essential in comparing quality of different software products. The overall quality of SharePoint software products was measured by the evaluation function, which parameters are based on the measures for each quality characteristics. Function is described in detail in following section.

6.5.3 Methods

6.5.3.1 Accuracy

\[
M_{acc} = \frac{C_c}{C_a} \times 100\%
\]

\(C_c\) - number of correct software components

\(C_a\) - number of all software components

Measure that is chosen to evaluate accuracy is percentage of the SharePoint software components, which did not undergo any corrections caused by inconsistency with specification (1). SharePoint software components include columns, content types, list templates, list instances, features and site definitions. Those entities are considered to be correct, when there is no need to change their data type (columns), contents (content types, list templates, features, site definitions), names and titles (all). High level of accuracy means that requirements were identified correctly, and most components were developed according specification. Accuracy was measured by analysing six performed projects. For each project, number of components, which were accepted without changes and number of all components, was determined based on the tasks that were performed during the project. If commentaries associated with
the task indicated, that as a part of it some corrections have been made, the corresponding component was not categorized as correct.

6.5.3.2 Appropriateness recognisability

\[ M_{app} = \frac{C_i}{C_a} \times 100\% \]

\( C_i \) - number of successfully identified components 
\( C_a \) - number of all components to identify  \hspace{1cm} (2)

Measure that is chosen to evaluate appropriateness recognisability (2) is percentage of successes in identifying components, which functionality or contents are stated in questions (e.g. “Which feature is responsible for installing content type X?”). High measure of recognisability means, that purpose of SharePoint components is easy to identify, thus little time is needed for their identification. Components are successfully identified, when their exact purpose is stated in given time. Ability to identify which element provides defined functionality is essential for testing, modifying or adding other functionalities, especially, when those tasks are not performed by developer who realized appropriate components. Appropriateness recognisability was measured by conducting test, during which subjects were asked several questions about components which realizes stated purpose. Each component has two representations – traditional and based on DSL. There were also two groups of subjects – first answered questions basing on traditional SharePoint development artefacts, while second had to analyse DSL diagrams.

6.5.3.3 Learnability

\[ M_{lrm} = \frac{C_d}{C_a} \times 100\% \]

\( C_d \) - number of components, which purpose was accurately defined
\( C_a \) - number of all components, which purpose was to define \hspace{1cm} (3)

Measure that is chosen to evaluate learnability (3) is percentage of successes in stating the purpose of given components (e.g. F1 is feature that instantiates list, and depends on the features that installs columns, content types - FX, and list templates necessary for list instances - FY). The higher rate of learnability is, the more understandable are particular artefacts, thus less time is needed for understand what their purpose is. The purpose of component is considered to be accurately defined, when all its contents attributes, contents and dependencies are included in that definition. Additionally, the purpose should be stated in given time. Learnability was measured by performing a test, during which subjects were asked about purpose of five components. Like in the previous case, subjects were divided into two groups, and answer those questions basing on DSL or traditional representations of components.
6.5.3.4 **Attractiveness**

\[ M_{atr} = \frac{\sum A_e}{A_a} \]

\( A_e \) – evaluation of particular attractiveness aspect \hspace{1cm} (4) \[ A_a \] – number of attractiveness aspects

Measure that is chosen to evaluate attractiveness (4) is mean evaluation of attractiveness aspects. Each aspect is represented by question stated in survey, which is source of data for measuring attractiveness. Each question is answered by assigning number between 1 and 10, to evaluate subjects’ compliance with statement included in it, where 1 means disagreement, and 10 – full agreement. There is equal number of corresponding aspects for software products created with and without SharePoint DSL.

6.5.3.5 **Ease of use**

\[ M_{eou} = \frac{\sum E_e}{E_a} \]

\( E_e \) – evaluation of particular ease of use aspect \hspace{1cm} (5) \[ E_a \] – number of all ease of use aspects

Measure that is chosen to evaluate ease of use (5) is mean evaluation of ease of use aspects. Like in case of attractiveness, aspects are represented by questions stated in survey, and they were answered by assigning number between 1 and 10. Interpretation of those answers is also similar - 1 means disagreement with the statement, while 10 – agreement. There is equal number of corresponding aspects for software products created with and without SharePoint DSL.

6.5.3.6 **Reusability**

\[ M_{reu} = \frac{C_r}{C_a} \times 100\% \]

\( C_r \) – number of reused components \hspace{1cm} (6) \[ C_a \] – number of all components

Measure that is chosen to evaluate reusability (6) is percentage SharePoint components that are used in more than one software products. High percentage of reused components means, that project was quiet similar to previously realized and less work is necessary in order to finish it. In situation, where project are dedicated to same segment of market, reusability should reach level higher than 10%. Set of this components consist of those, which were developed for the purpose particular project, and prove to be useful in another. Those components are in most cases features that deploy SharePoint data structures, and features that install new functionality, such as event handlers. Reusability was measured in six conducted projects, by analysing project-related artefacts such as source code, DSL model, CAML definitions and documentation.
6.5.3.7 **Changeability**

\[ M_{chn} = \frac{\sum C_c}{C_a} \]

\( C_c \) – time spent on introducing changes to particular component  \( (7) \)
\( C_a \) – number of all modified components

Measure that is chosen to evaluate changeability (7) is mean time spent on modifying components. It is quite obvious, that time spent on modifications should be as short as it is possible. However, in real life situations, the fact that it does not exceed time spent on initial development is sufficient. Changes in SharePoint components may be introduced in many different ways, and complexity of those changes also varies. However, there is group of modifications, which occur more frequent than others (see Error! Reference source not found.). That is why measure of changeability for different projects, which were sources of data, might be evaluated and compared.

6.5.3.8 **Modification stability**

\[ M_{mstab} = \frac{\sum C_f}{C_a} \]

\( C_f \) – number of components affected by particular modification  \( (8) \)
\( C_a \) – number of modified components

Measure that is chosen to evaluate reusability (8) is mean number of SharePoint components that must be manually modified after altering other components, in order for the product to remain stable. The lesser is the measure of modification stability, the smaller amount of modification is necessary. Software product is considered to remain stable, when its behaviour after introducing changes is the same as before that. Changes that may be made in particular components, along with types of components which potentially may be affected by those changes are listed in Error! Reference source not found..

<table>
<thead>
<tr>
<th>Modified component</th>
<th>Type of modification</th>
<th>Types of components affected</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Column</strong></td>
<td>Changing data type</td>
<td>List Template</td>
</tr>
<tr>
<td></td>
<td>Changing properties</td>
<td>Content Type, List Template</td>
</tr>
<tr>
<td><strong>Content type</strong></td>
<td>Adding or removing</td>
<td>Content Type, List Template</td>
</tr>
<tr>
<td></td>
<td>Columns</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Replacing parent</td>
<td>Content Type, List Template</td>
</tr>
<tr>
<td></td>
<td>Content Type</td>
<td></td>
</tr>
<tr>
<td><strong>List Template</strong></td>
<td>Adding or removing</td>
<td>List Template</td>
</tr>
<tr>
<td></td>
<td>Content Types</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Changing properties</td>
<td>List Instance</td>
</tr>
<tr>
<td><strong>List Instance</strong></td>
<td>Changing Url</td>
<td>Column</td>
</tr>
<tr>
<td><strong>Feature</strong></td>
<td>Adding or removing</td>
<td>Feature</td>
</tr>
<tr>
<td></td>
<td>Content Types, Lists</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Templates and Instances, Columns</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Changing sequence</td>
<td>Feature, Site Definition</td>
</tr>
</tbody>
</table>
6.5.3.9 Overall Quality

First step to produce valuable input for this function was to calculate mean value for each characteristic, because in all cases several objects, (such as data associated with projects, results of survey or tests performed among many developers) were analysed in order to evaluate certain aspect of quality.

After calculating mean values of all characteristics, they had to be transformed due to the fact that initial measures were taken in several different scales. Additionally, semantics of each measure for quality characteristics had to be taken into consideration, because in some cases higher values have negative meaning, while in others - positive (e.g. time spent on introducing changes versus level of satisfaction in working with tool). Without proper adjustments, reasonable quality evaluation would be impossible – one characteristic expressed in value measured in $1 – 10$ scale would level those measured in scale $0 – 1$. For that reason, three transformations were introduced and applied for mean values of characteristics. Transformations were designed in a way, which makes their outcome as close to $0 – 1$ scale as it is possible. The reason for that is the fact that most of the measures are already close to that scale.

In case of values expressed in percentages (accuracy, appropriateness recognisability, learnability), the property of percentage, which states that percentage sign can be treated as equivalent of number 0,01, was utilized (9).

\[
V_{PA} = V_p \times 0,01
\]
\[
V_{PA} \quad \text{adjusted value}
\]
\[
V_p \quad \text{unadjusted value expressed in percentages}
\]

Measures, which were outcome of performed survey, and which possible values were in range between 0 and 10 (attractiveness, ease of use) were adjusted by dividing those values by 10 (10).

\[
V_{SA} = V_s / 10
\]
\[
V_{SA} \quad \text{adjusted value}
\]
\[
V_s \quad \text{unadjusted value, which range was between 0 and 10}
\]

In case of changeability and modification stability, lesser values have to be interpret as better than higher. Additionally, in both cases their multiplicative inverses are close to desired range. For those reasons and for the purpose of this work, multiplicative inverse (11) is sufficient transformation to adjust changeability and modification stability measures – it makes adjusted measures of given characteristics close to desired scale, and influence overall quality in desired manner (lesser values of raw values have positive influence on the outcome, while higher – negative).

\[
V_{nA} = 1 / V_n
\]
\[
V_{nA} \quad \text{adjusted value}
\]
\[
V_n \quad \text{unadjusted value, which has negative meaning}
\]
Values presented in Error! Reference source not found., were input for function, which was used to evaluate overall quality of SharePoint software products (12). Function is weighted mean of values measured for each quality subcharacteristic.

\[
Q = \frac{\sum Q_w \cdot Q_v}{\sum Q_w}
\]

\(Q_w\) – weight of quality characteristic
\(Q_v\) – value of quality characteristic

(12)

In this work, measures of characteristic have equal importance, thus weigh assigned to each measure is the same, and equal 1. Nonetheless, proposed weights are dynamic, so it is possible differentiate them in order to fit the function to more specific needs and preferences. For example, when evaluation should put more emphasis on aspects of the language associated with its usage, weighs for aspects such as attractiveness and ease of use should be increased.

6.5.4 Results

6.5.4.1 Accuracy

Results of evaluation of accuracy for projects developed with and without SharePoint DSL are shown on Error! Reference source not found..

![Figure 34. Comparison of accuracy](image)

6.5.4.2 Appropriateness recognisability

The result of a test is presented on Error! Reference source not found..
6.5.4.3 Learnability

Result of test is shown on Error! Reference source not found..

6.5.4.4 Attractiveness

Results of the survey, as a comparison of evaluation of attractiveness of software products developed with and without SharePoint DSL, are shown on Error! Reference source not found..
6.5.4.5 Ease of use

Results of survey (Error! Reference source not found.) are presented in similar manner as those for attractiveness.

6.5.4.6 Reusability

Outcome of analysis of artefacts for each project is shown on Error! Reference source not found..
6.5.4.7 Changeability

Error! Reference source not found. presents outcome of evaluation of changeability for each project.

6.5.4.8 Modification stability

Modification stability was measured by analysing artefacts associated with six performed project, and outcome of this analysis is shown on Error! Reference source not found..
6.5.4.9 Overall Quality

<table>
<thead>
<tr>
<th>Quality Characteristic</th>
<th>Measure for products developed without DSL</th>
<th>Measure for products developed with DSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy (Acc)</td>
<td>0,67</td>
<td>0,85</td>
</tr>
<tr>
<td>Appropriateness recognisability (App)</td>
<td>0,72</td>
<td>0,96</td>
</tr>
<tr>
<td>Learnability (Lrn)</td>
<td>0,72</td>
<td>0,96</td>
</tr>
<tr>
<td>Attractiveness (Atr)</td>
<td>0,28</td>
<td>0,90</td>
</tr>
<tr>
<td>Ease of use (Eou)</td>
<td>0,30</td>
<td>0,86</td>
</tr>
<tr>
<td>Reusability (Reu)</td>
<td>0,07</td>
<td>0,20</td>
</tr>
<tr>
<td>Changeability (Chn)</td>
<td>0,55</td>
<td>1,50</td>
</tr>
<tr>
<td>Modification stability (Mstab)</td>
<td>0,22</td>
<td>0,90</td>
</tr>
</tbody>
</table>

Table 8. Adjusted values of quality characteristics

The result of applying formula described in section 6.5.3.9. to results of evaluation of products developed with and without support of SharePoint DSL is shown on Table 9.

<table>
<thead>
<tr>
<th>Quality of software products developed without SharePoint DSL</th>
<th>Quality of software products developed with SharePoint DSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,44</td>
<td>0,89</td>
</tr>
</tbody>
</table>

Table 9. Comparison of evaluation of software products quality
6.5.5 Analysis

6.5.5.1 Accuracy

Inconsistencies between SharePoint software components and specification are caused by assigning wrong names to those entities or imprecise definition of their properties and contents, such as incomplete set of columns in content type, or excessive number of content types in list template. The reason for increased accuracy in projects, which were developed with help of DSL, is the fact that they were much easier to verify during early stages of development. Components, which were developed by traditional means, may be verified by reviewing XML definitions along with resource files, which is time consuming and cannot be done by customer. On the other hand, verification of DSL diagram is much easier and effective - properties and contents of SharePoint data structures are easier to compare with documentation, and may be verified by customer. For that reason, accuracy for projects developed with SharePoint DSL achieved level close to 90%, whereas before introducing it into development language, it oscillates between 60% and 70%.

6.5.5.2 Appropriateness recognisability

Recognition of functionality and contents of each SharePoint component is made by analysis of their definitions. XML definitions of SharePoint components are difficult to read, especially when some of them contains references to others. Unfortunately, those relationships have to be identified on many occasions, such as introducing changes or verifying existing functionality. The easiest way to find given component is to use Visual Studio IDE search mechanisms, but it still takes some time and is not intuitive – in most cases searched phrase is GUID of component, and a set of files, which contain this string, is search result. Graphical representations of links between SharePoint components allow almost instantaneous identification of such relationships. Additionally, relationships such as hierarchies and inclusions are illustrated in distinctive, yet more convenient way. For that reasons recognisability in projects developed with SharePoint DSL reached level close to 100%, when average in project developed in traditional way was 70%.

6.5.5.3 Learnability

There are many situations, when once developed components have to be modified, such as introducing changes or performing tests. In some of those cases, actions are made by programmers, who have not developed those components, so they have to identify the purpose of each component by themselves. In order to assist them in this task, SharePoint DSL diagram provides notation more readable than CAML definitions, so the purpose of each component may be identified faster and more accurate. Learnability of applications developed with SharePoint DSL reached level close to 100%, when for those developed with standard methods – less than 80%. Additionally, SharePoint DSL diagrams are great assets in teaching new developers about SharePoint components by “learn by example” method. Possible types of relationships between components, and assistance in specifying their properties are very helpful for novice developers, and diagrams are valuable source of domain knowledge about SharePoint development.
6.5.5.4 **Attractiveness**

The result of conducted survey is strongly in favour of DSL. Software products developed without support from SharePoint DSL rarely exceeded level of 3, which means that they are generally considered to be unattractive. On the other hand, software products developed with DSL were never rated below 8. The attractiveness of software products developed with DSL is caused by improvements, which were brought by it into development process, such as code generation, graphical notation and validation. The unattractiveness of software products developed without DSL is caused by necessity of work with large and difficult to read XML files.

6.5.5.5 **Ease of use**

Like in the case of attractiveness, results of the survey show large divergence between software products developed with and without DSL – first group was rated much higher than second. The reasons are also similar – DSL diagrams are much easier to use than XML definitions, because they provide access to all SharePoint components from one place, and those components are presented in more convenient and readable manner, while distributed definitions of those components, which were developed using standard techniques are much more difficult to handle.

6.5.5.6 **Reusability**

Before SharePoint DSL was introduced into software development process, degree of components reusability was not satisfactory. Despite the fact, that all projects provide similar functionality, and were aimed for the same market, most of components were created from scratch. Main reason for that situation was lack of knowledge about existence and capabilities of assets developed in previous projects. Although contents and functions of all features were pretty well documented, access to assets was difficult. They are mainly xml definitions of SharePoint data structures, which are difficult to read and interpret. On the other hand, introduction of SharePoint DSL increased level of reusability, by introducing diagrams, which illustrates all components which are parts of particular project. Graphical representation of features and their contents, as well as relationships between SharePoint data structures makes them easier to analyse, and thanks to that they were reused more often. First three projects were developed without SharePoint DSL, and number of features reused from previous projects does not exceeded 10%. After introducing SharePoint DSL into development process, number of assets, which are reused from previous projects, came close to 25%.

6.5.5.7 **Changeability**

Mean time of introducing changes in projects, which were developed using standard techniques, was close to 2 man-hours. This situation was not satisfactory, because time spent on modifying components was, in most cases, comparable with time spent on their initial development. The main reason for that was complexity of XML definitions of those components, which sometimes makes their modification more difficult than initial development. During modifications, appropriate segments of XML file must be first found, and then modified, whereas during initial development, there is no need for searching for them. In projects, which were developed with
SharePoint DSL, mean time spent on introducing changes was less than 1 man-hour. DSL diagrams prove to be more manageable and changeable, mainly because they expose only relevant information about SharePoint components. This was the reason for reducing time spent on identifying attributes which must be changed to minimum. Additionally, DSL diagrams provide easy access to all definitions of SharePoint components, while in standard approach to SharePoint development, those definitions were distributed among many separate files. However, changes made through DSL bring additional time, which is caused by model transformation, after which actual artefacts with definitions are generated.

6.5.5.8 Modification stability

For projects, which were developed without DSL, most of the changes necessary for keeping them stable were made manually. Additionally, those changes had to be done in multiple places, especially when source of modification was Content Type. For that reason, modification stability was quite high - 4.5 - which means that every change introduced into one component, requires additional changes in 4 or 5 other components. Considering the fact, that changes occur frequently, and are usually time consuming, keeping software products stable was expensive, but necessary part of development process. Introduction of SharePoint DSL reduced number of components, which must be manually modified to 1. The reason is that most of necessary changes are done automatically during model transformation, so there is no need to make them manually. For example, replacing parent Content Type requires no additional work, because identifiers for Content Types are generated and updated automatically. Automation of propagation of changes greatly reduces amount of work associated with software product modifications.

6.5.5.9 Overall Quality

The result of evaluating software products prove, that those developed with assist of SharePoint DSL have much higher (almost two times) quality than those developed without its support. It was foreseeable outcome, considering the fact that in all cases adjusted measures were higher in favour of those made for DSL software products. Moreover, some of the measures were 3 or even 4 times higher for software products made with DSL. First characteristic corresponding with those measures is Operability, which means that software products made with assistance of SharePoint DSL are easier to learn, more understandable and more attractive. The reason for that was introducing graphical notation, which is more convenient way of presenting information than XML-based text files. Second characteristic is Maintainability – modifications are easier to make in software products made with SharePoint DSL, mainly due to storing information on one diagram, which otherwise are distributed among multiple files in different locations.
6.6 Validity threats

Validity threats are possible sources of making statement false or greatly inaccurate. This section describes validity threats to the evaluation of quality of SharePoint software products made with and without SharePoint DSL tool. All definitions and categorizations of validity threat were based on works [39] and [40].

The following sections will be based on two basic terms:
- independent variable – variable that represents value being changed during the course of the experiment
- dependent variable – variable that represents value changed by the manipulation of the independent variable

6.6.1 Internal Validity

First set of validity threats is can affect so called internal validity, which can be defined as extent to which independent variable produced the observed effect. Internal validity is achieved, when effect on dependent variable is made only due to variation in independent variable.

6.6.1.1 Confounding

Confounding occurs, when there is variable different than independent variable, that influences dependent variable, and systematically varies with variation of independent variable. In case of this work, there are two threats of this kind identified:
- increase in developers skills while introducing SharePoint DSL into development process – this threat is fully described in section 6.5.1.2
- fact, that SharePoint DSL was made by some of the test subjects (very close to the “judge in his own cause” situation) – it can affect those measures, which are related to subjective subcharacteristics such as ease of use, attractiveness;

6.6.1.2 History

History is the can affect the internal validity, when events outside the area of the research, which occurred between changes of independent variable, might influence value of the dependent variable. In case of this study, such event was change in the model of software development. To some point, products delivered by IT-Dev were oriented to particular client. Since then, development and sales model evolved, in order to produce customizable products rather than solutions dedicated for one, particular use. For that reason, some of the later projects, which are important source of the data, were made which more emphasis on reusability, thus it might affected such subcharacteristics as reusability, changeability and modification stability.

6.6.1.3 Maturation

Maturation is the threat caused by the changes of the test subjects with passage of time. Maturation is the largest threat to validity of measures, which were made basing on the data collected from performed projects. All of those projects were made by the same company and by similar developer teams. Additionally, most of the projects
made without SharePoint DSL were made earlier than those made with the tool. Those are the reason to suspect, that some of the improvements of the quality might be caused by improvement of skills of the developers. This threat could affect measures for appropriateness recognisability, accuracy, learnability (more experience, intuition, learned habits).

6.6.2 External Validity

Second group of threats can affect external validity, which is in general ability to generalize the findings of the research from exemplars to entire class of populations or conditions. This group of threats is caused by the fact, that experiments which involve human participants employ small samples with quite similar features (e.g. students from the same university).

6.6.2.1 Aptitude-Treatment Interaction

This threat is caused by the fact, that the sample may have certain set of features that interact with the independent variable. In such case, test made on other samples will provide different outcome. In this work, Aptitude-Treatment Interaction is the main threat to its validity. The main reason is the fact, that the sample group of developers, which had great influence on all sources of data (projects, survey, and test – all described in subsections of section 6.3.1), had experience with the development process with and without SharePoint DSL. The knowledge of the tool could affect measures for appropriateness recognisability. The reason is also partial involvement of the sample group in design, creation and development of the tool, which might influence measures for attractiveness an ease of use in the same manner as confounding.

6.7 Summary

Usefulness of proposed DSL was admitted, by evaluating its influence on various software product aspects, and comparing it with evaluation made for software products made without its support. Evaluation was made by applying formal method of measuring software quality, developed by International Organisation for Standardisation - building a quality model, and measuring quality basing on model’s components. The method was tuned, adjusted and applied to those aspects of software products, which could be influenced by introducing SharePoint DSL. Each subcharacteristic was evaluated by applying defined measuring method. Next step was to compare evaluation outcome for products developed with and without support from DSL. In all cases, introducing SharePoint DSL into development process results in increasing software quality in appropriate areas. It confirms that DSL indeed has positive impact on development process and software product, and effort made for DSL development was profitable. However, there are several threats to validity of presented research, and their elimination must considered as most important part of future work.
7 Summary

7.1 Conclusions

This work presented application of Domain-Specific Languages in solving problems from area of Enterprise Portals development. Its main aim, which was increasing quality of software products made on SharePoint platform, was achieved by developing Domain-Specific Language dedicated for modelling certain aspects Enterprise Portals. The language development process was conducted with respect to guidelines and recommendations that can be found in literature. Nevertheless, those guidelines did not cover all aspects of DSL development, so in some areas process had to be improved or even invented.

Proposed language utilizes various mechanisms, which reduce developers work and increase product quality. The most important aspect of the language is its notation – instead of working with XML-based definitions, which are distributed among multiple files, SharePoint DSL offers graphical notation. Its main advantages are ability to store all definitions in one place, and graphical representation of relationships between SharePoint entities. Such kind of notation makes possible involving customers in development process. Nevertheless, ability to represent SharePoint entities would not be useful without mechanisms for generating actual definitions. For that reason, SharePoint DSL utilizes artefact generation mechanisms, which enables creating XML definitions of SharePoint entities and code, which assist in further development.

SharePoint DSL is useful not only for support development. It is also storage for domain knowledge, which can be used to teach inexperienced developers about SharePoint data structures and deployment packages. It is also valuable asset for software architects, who may design arrangement of particular components among sites or features. Advantage of diagrams is also their readability and understandability to layman. Thanks to that, some of the components may be verified in early stages of development process.

In order to verify the usefulness of the language, software products developed with its support have been compared with those developed using traditional approach. This comparison was based on evaluation of products quality. In order to do that, quality model for SharePoint software products was developed. Model is based on upcoming standard for evaluating software quality – ISO/EIC 25010. Subcharacteristics, which are contents of proposed model, were selected with respect to areas, in which changes caused by introducing DSL might appear. To provide more complex evaluation of products’ quality, thus enable their comparison in other perspectives, quality model should be expanded with other subcharacteristics.

SharePoint DSL proved to be crucial asset in development of SharePoint-based portals, and became most important tool for supporting software development in IT-Dev company. Thanks to that tool, IT-Dev has been awarded by Microsoft as a Partner of the Year in category “Information Worker Solutions, Portals and Collaboration”. [19] Currently tool is being prepared to be sold outside company.
7.2 Research questions revisited

Q1. What has been proposed in literature about DSLs?

Domain-Specific Language became more and more popular in recent years, mostly because of their usefulness in development process. Publications such as [36] are valuable sources of information about DSLs, especially when searching for examples of their successful development. Besides articles, there are also entire books about DSLs, such as [3], which describes DSL development in Visual Studio IDE. There are however lack of more complex publications, in which knowledge and experience associated with DSL would be gathered and systemized. Most information about DSLs were gathered in chapter 3.

Q2. How the process of development of DSL is conducted?

Domain-Specific Languages are relatively new concept, so there is not much information and support in area of their development. There are however some guidelines for their development, which determine shape of the DSL development process, and possible approaches to each phase of this process. Those techniques were presented in 3.5.

Q3. How to improve the quality of EP on SharePoint platform?

Number of problems, which surface during development of portals on SharePoint platform indicates, that there are lot of possibilities of improving that process and quality of software products. Building methods for automatic artefact generation, employing customers into development process or providing mechanism for domain rules validation are those, which were included in this work.

Q4. What are the key aspects of EP that are suitable for creating DSL?

Most crucial aspects of Enterprise Portals, which needs support, are declarative definitions of data structures and deployment packages. Those entities, used to be defined via complex XML definitions, which were difficult to manage and interpret by developers. SharePoint DSL provided graphical notation, which allows more intuitive representations of SharePoint entities and relationships between them.

Q5. How the usage of DSL during development of EP affects the products’ quality?

Including SharePoint DSL into development process had an impact on quality of developed software. By providing mechanisms, which reduce performing repetitive task by programmers, and increasing engagement of architecture and customers in development process, it was possible to focus on aspects such as reusability, changeability, modification stability or ease of use. In order to measure those changes, quality model for evaluating quality of SharePoint software products was developed. After measuring each of quality characteristics, and applying those measures in evaluation function, differences between software products developed with and without support of SharePoint DSL were determined. According to those results,
SharePoint DSL increased internal quality of SharePoint software products about two times. Such increase made worthy an effort made in order to develop that language.

7.3 Future work

Most important part of future work should be eliminating some of the threats to validity of the research. Most of those threats were caused by conducting research in closed environment. All sources of data (projects, survey, test) came from one company. On the other hand, those threats are easy to eliminate, by widening sources of data, which could include projects made by other companies or surveys and test conducted within SharePoint developers from outside company.

Language and supporting tool were designed and developed for with Visual Studio 2005 IDE. The reason for that is the fact, that company for which language was developed, at this time had access only to that version of IDE. However, since licenses for Visual Studio 2008 have been purchased, there are on-going works, which aim is migration of tool. On the other hand, Microsoft has announced evolution of DSL Tools in Visual Studio 2010. DSL SharePoint development team is tracking all those novelties, and selects those which could improve the language.

Second area, in which language could be improved, is its functionality. Language proposed in this work is focused only on data structures and deployment packages, and even in those areas it does not allow modelling them in full extent. For example, language should be enhanced with ability to define custom data types, or at least allow working with those types. There are also other areas, which could be included in language. First of them is defining search scopes – sets of rules, which defines elements which should be indexed and made available to be found. Those rules are often based on the values stored in particular columns. Second area is security – SharePoint allows defining custom levels of permissions, and associating those permissions with groups of users and SharePoint entities such as lists or particular items on those lists. Those two mechanisms are closely related to entities, which already can be modelled with proposed language. Additionally, their definitions are also stored within XML files. Those two factors make them candidates for further improvements of SharePoint DSL.

Next part of this work, which might be improved, is quality model, which was used to evaluate quality of software products. Characteristics, which were chosen as contents of this model, where selected basing on the areas, in which changes caused by introducing DSL might occur. However, it is possible that changes unintentionally occur in other areas of software products. In order to verify this quality model may be expanded with other subcharacteristics. There is also possibility of improving methods of evaluation those subcharacteristics, which are already part of the model, by defining additional measures of them.
8 Bibliography


[32] Christopher Shilakes and Julie Tylman, Enterprise Information Portals. New York:

[33] Diomidis Spinellis, "Notable design patterns for Domain-Specific Languages," The

[34] Scott Thilbaut, Renaud Marlet, and Charles Consel, "Domain-Specific Languages:
From Design to Implementation Application to Video Device Drivers Generation,"

[35] Arie van Deursen and Paul Klint, "Little Languages: Little Maitenance?," Journal of

[36] Arie van Deursen, Paul Klint, and Joost Visser, "Domain-Specific Languages: An


[38] Colin White, The Enterprise Information Portal Marketplace. Morgan Hill: Database

[39] William Shadish, Thomas Cook, and Donald Campbell, Experimental and quasi-

Appendix A: Table of abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CAML</td>
<td>Collaborative Application Mark-up Language</td>
</tr>
<tr>
<td>CMS</td>
<td>Content Management System</td>
</tr>
<tr>
<td>CRM</td>
<td>Customer Relationships Management</td>
</tr>
<tr>
<td>DMS</td>
<td>Document Management System</td>
</tr>
<tr>
<td>DSL</td>
<td>Domain-Specific Language</td>
</tr>
<tr>
<td>EIP</td>
<td>Enterprise Information Portal</td>
</tr>
<tr>
<td>EP</td>
<td>Enterprise Portal</td>
</tr>
<tr>
<td>ERP</td>
<td>Enterprise Resource Planning</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>GPL</td>
<td>General Purpose Language</td>
</tr>
<tr>
<td>GUID</td>
<td>Global Unique Identifier</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>HTTPS</td>
<td>Hypertext Transfer Protocol Secure</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IIS</td>
<td>Internet Information Services</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>PWA</td>
<td>Microsoft Project Web Access</td>
</tr>
<tr>
<td>SaaS</td>
<td>Software-as-a-Service</td>
</tr>
<tr>
<td>SMTP</td>
<td>Simple Mail Transfer Protocol</td>
</tr>
<tr>
<td>SPD</td>
<td>Microsoft Office SharePoint Designer</td>
</tr>
<tr>
<td>SPL</td>
<td>Software Product Line</td>
</tr>
<tr>
<td>MDA</td>
<td>Model-Driven Architecture</td>
</tr>
<tr>
<td>MOSS</td>
<td>Microsoft Office SharePoint Server</td>
</tr>
<tr>
<td>MSP</td>
<td>Microsoft Office Project</td>
</tr>
<tr>
<td>MPS</td>
<td>Microsoft Office Project Server</td>
</tr>
<tr>
<td>MVC</td>
<td>Model-view-controller</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
<tr>
<td>VB</td>
<td>Visual Basic</td>
</tr>
<tr>
<td>VS</td>
<td>Microsoft Visual Studio</td>
</tr>
<tr>
<td>WIX</td>
<td>Windows Installer XML</td>
</tr>
<tr>
<td>WSS</td>
<td>Windows SharePoint Services</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Mark-up Language</td>
</tr>
<tr>
<td>XSLT</td>
<td>Extensible Stylesheet Language Transformation</td>
</tr>
</tbody>
</table>
Appendix B: Survey

This section contains questions, which were included in survey performed among SharePoint developers. First section contains questions associated with attractiveness, while second – related to ease of use.

Attractiveness Survey

A1: Indicate degree of convenience of graphical notations (1 – not convenient, 10 – convenient):
   1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10 ☐
A2: Indicate degree of convenience of textual notation (1 – not convenient, 10 – convenient):
   1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10 ☐
A3: Indicate degree of satisfaction for work with DSL diagrams (1 – very dissatisfied, 10 – very satisfied):
   1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10 ☐
A4: Indicate degree of satisfaction for work XML definitions (1 – very dissatisfied, 10 – very satisfied):
   1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10 ☐
A5: Indicate degree of comfort for work with DSL diagrams (1 – very uncomfortable, 10 – very comfortable):
   1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10 ☐
A6: Indicate degree of comfort for work XML definitions (1 – very uncomfortable, 10 – very comfortable):
   1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10 ☐
A7: Indicate degree of pleasure for work with DSL diagrams (1 – very displeased, 10 – pleased):
   1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10 ☐
A8: Indicate degree of pleasure for work XML definitions (1 – very displeased, 10 – pleased):
   1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10 ☐
A9: Indicate degree of attractiveness of graphical notations (1 – unattractive, 10 – attractive):
   1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10 ☐
A10: Indicate degree of attractiveness of textual notations (1 – unattractive, 10 – attractive):
    1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10 ☐
Ease of Use Survey

E1: Indicate level of difficulty of finding information on DSL diagram (1 – impossible, 10 – very easy):
1 □  2 □  3 □  4 □  5 □  6 □  7 □  8 □  9 □  10 □

E2: Indicate level of difficulty of finding information in XML definitions (1 – impossible, 10 – very easy):
1 □  2 □  3 □  4 □  5 □  6 □  7 □  8 □  9 □  10 □

E3: Indicate the level of understanding of DSL diagrams (1 – not understandable, 10 – understandable):
1 □  2 □  3 □  4 □  5 □  6 □  7 □  8 □  9 □  10 □

E4: Indicate the level of understanding of XML definitions (1 – not understandable, 10 – understandable):
1 □  2 □  3 □  4 □  5 □  6 □  7 □  8 □  9 □  10 □

E5: Indicate degree, to which working with DSL diagrams assists in achieving your goal (1 – it disturbs 10 – it is very helpful):
1 □  2 □  3 □  4 □  5 □  6 □  7 □  8 □  9 □  10 □

E6: Indicate degree, to which working with XML definitions assists in achieving your goal (1 – it disturbs 10 – it is very helpful):
1 □  2 □  3 □  4 □  5 □  6 □  7 □  8 □  9 □  10 □

E7: Indicate level of manageability of DSL diagrams (1 – unmanageable, 10 – very manageable):
1 □  2 □  3 □  4 □  5 □  6 □  7 □  8 □  9 □  10 □

E8: Indicate level of manageability of XML definitions (1 – unmanageable, 10 – very manageable):
1 □  2 □  3 □  4 □  5 □  6 □  7 □  8 □  9 □  10 □

E9: Indicate degree, to which you consider DSL diagrams to be easy to work with (1 – very difficult to work with, 10 – very easy to work with):
1 □  2 □  3 □  4 □  5 □  6 □  7 □  8 □  9 □  10 □

E10: Indicate degree, to which you consider XML definitions to be easy to work with (1 – very difficult to work with, 10 – very easy to work with):
1 □  2 □  3 □  4 □  5 □  6 □  7 □  8 □  9 □  10 □