Improvement of Software Development Process with use of Template Generator

Ivan Grbavac
Sveučilište u Dubrovniku
ivan.grbavac@unidu.hr

***Abstract*— In today's society, where access to the technology is becoming a basic human need, the need for software applications and developers is rapidly increasing while the labor market does not meet the needs of entrepreneurs. Workload of software companies is constantly increasing, as they not only have to develop new applications, but must also maintain the existing ones. In order to respond to the growing demands, code generators, tools which automatically generate production-ready source code based on given template, have been developed. But implementation of code generators and templates is very hard and time consuming job which requires a wide knowledge of metalanguages of which templates are usually made. Also, structure of templates sometimes can be very complex. During process of code maintenance, development of new modules, or simply development of new applications based on old ones, developer is faced with a choice: to use code generators or code manually. In order to facilitate developers' tasks this paper presents concept of template generator and proposes its design. It is a new tool which could be used to discover crosscutting concerns in existing source code and automatically generate code templates leaned to aspect oriented paradigm. Use of template generator would provide well-structured generated code, faster and cheaper application development and maintenance, and would eliminate a requirement for a wide knowledge of template metalanguages.**

***Keywords*— template generator, CASE tools, code generator, crosscutting concerns.**

I. INTRODUCTION

THIS modern information era society imposes great necessity of various software applications which now have an increasingly important impact on human life. To make up that demands enterprises have to make more applications in shorter time frames, but also have to maintain existing software [1]. In addition, they have to promptly respond to the business changes and do so in an appropriate manner [2]. Due to mentioned reasons it is not surprising that some analysts report that the failure rate may exceed 50% of all projects [3]. In order to respond to the growing challenges various Computer Assisted Software Engineering (CASE) tools have been developed. Also, in the eighties of the past century the idea of code generators, programs that helps to write production-ready parts of the source code, was born.

With today's complex code-intensive frameworks, such as Java 2 Enterprise Edition (J2EE), Microsoft's .NET, and Microsoft Foundation Classes (MFC), it's becoming more important that programmers' skills are used to build and/or use code generators, which assist in building applications and downgrading time and expenses needed for accomplishing projects [4]. In addition to the financial benefits, code generators preserve identical code structure and way of coding in every generated file, which makes next cycle in software life easier.

But building new applications is not the only developers' job. Software companies may also reuse and improve existing software which was developed by other companies or by former employees. If the source code, in specific project, was not well-structured or was not made by the help of code generator, its maintenance becomes time consuming and more human resources are allocated to assist in software lifecycle. The problem becomes greater when the cycle comes to a point without a support for a framework in use or when programming language becomes obsolete. Then it is necessary to retype the entire project into a new development environment. Such events may occur also in lifecycle of software developed with code generators, but the change to another language and other development environment goes faster because most of the work is related only to templates changing. Process is even faster if the templates are made with language free data model, than it is necessary only to load new description of target language and generate new source code.

In order to facilitate developers' tasks this paper describes a proposal for improving reuse and lifecycle of software using proposed template generator. Its task is to analyze the original hand-written or generated code, to discover crosscutting concerns, suggest proper code structure and turn it into a template that could automatically generate source code and consequently speed up the process of developing and maintaining software. It would also eliminate a requirement for developers' wide knowledge of template metalanguages.

Following sections describe template generator role in software lifecycle and its design. Process of source code mining and converting it to template written in metalanguage is also described. Example of transformation C# source code to C++ code, using templates, is given.

II. Software Development Process

Traditional software development process consists of analysis, design, implementation, and testing [5], [6]. In a case when developers didn't use code generators or their use was minimal, produced application source code can be categorized into three classes (Fig. 1.) [7], [8]:

- common class, common to the most of developed applications in a software enterprise, where applications consist only of common libraries,
- resemble class, similar to common, but made without use of code generators, which should be handmade for every application and different by writing style depending on developer,
- different, application-specific, class that depends completely on application, also handwritten and cannot be automatically generated.

Without use of any code generator tools, only common libraries are reused by developers on their own risks to reduce the coding cost. Only the code in the resemble class might be completely automatically generated by code generators and its proper structure can increase the software reuse and can reduce the coding cost. Of course, some parts of different class might be also generated, but those parts require programmers' involvement. It becomes obvious that the biggest improvement of software lifecycle can be achieved by optimizing or automating the development of the middle class. That improvement might save even more time and money on maintenance or further development of existing projects, as described in following examples.

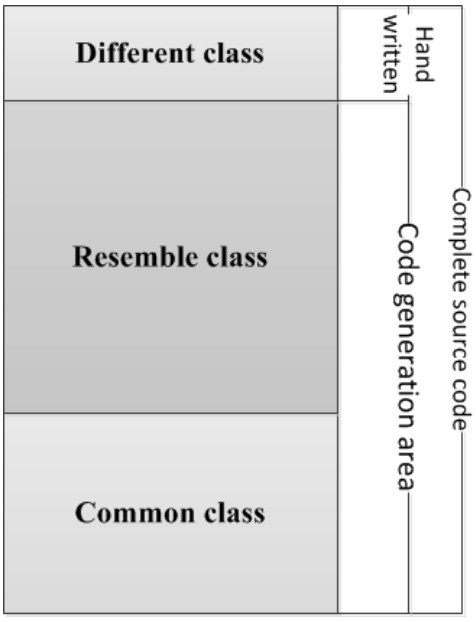

Fig. 1. Application Code Classification

During long development process applications may encounter problems, for example, application development can be put aside and subsequently continued but without the developers who started development. In this case, it is necessary first to examine the source code previous employees made, which was not, in many cases, properly structured and which varies depending on the programming style. Of course, it is a time consuming process. A similar problem may occur after the end of the software development and deployment. Then, the software manufacturer has to work on maintenance, further development and improvement of the system, usually in way as indicated in Fig. 2. In such continuation of the lifecycle it is expected not only that one developer will be replaced by other, but because of the length of software's life it is expected that several employees will work on maintenance of a particular part of code during its lifecycle.

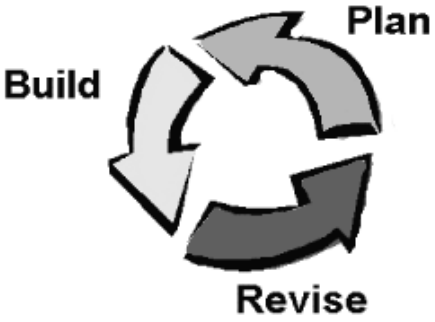

Fig. 2. Software lifecycle

A. *Template Generator Role in Software Development Process*

As previously indicated, the software lifecycle and maintenance, if software was not developed with the help of code generators, can be expensive and time-consuming. Development of new application requires analysis, design, coding and testing while after-birth system engineering processes require time-consuming and in-depth understanding of each system component in order to refactor it, to make it evolve, to migrate it to a new platform or to integrate it into a larger system [9]. As shown in Fig. 1, a specific part of the code is common or resemble to all applications and modules while a small piece of code varies depending on the application role. These two basic parts of the software, resemble and common class, due to its similarity have the potential to, with the help of text mining processes, contain certain patterns which can be converted into templates for future generation of source code. In this way the module, layer or library could be turned into a set of templates that could, with minor modifications via a graphical interface, generate structured production-ready source code. Those parameterized templates could be used in the further development of the software.

Of course, one part of the code can never be turned into template and parameterized as this code performs a specific function in a particular application. Template generator should indicate the existence of such code and protect it from

potential modification [10]. By doing all mentioned, following effects might be achieved:

- Reduced development time of new functionality or modules: if patterns, by which old parts of the application were developed, exist there is a likelihood that at least some parts of the code could be turned into templates and that new source code could be produced automatically.
- Reduced number of errors: templates, generated from existing code, are used during new development and the possibility of errors is minimized. If template contains an error, it can be easily fixed.
- Easier maintenance and better performance: in case when existing modules need to be amended, it is enough to change the template or parameters and again generate modules, same as in conventional template code generation.
- Code reuse: if software enterprise works on similar projects, templates used in one project might be used in others. So the whole cycle of analysis, design, implementation and testing is being shortened once again.

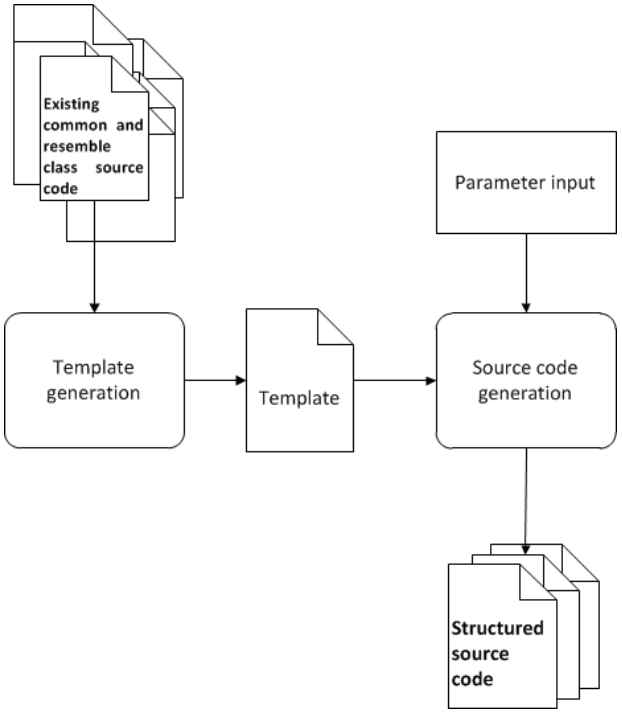

Fig. 3. Code maintenance with template generator

Another benefit of using template generator and code generator is production of well-structured source code as shown in Fig. 3. Such code is easy to maintain even manually.

III. Templates Generator Model

A. The Initial Model

Development of suggested model started with simple problem: creation of model for converting one source code file into template [11]. During the research of similar works [12], [13], [14], simple model was adopted as shown in Fig. 4.

In presented model, source code file is parameterized, and parameters from it are extracted. Initial parameter values are put in XML format and saved in the database while parameterized source code goes for further processing. Parameterization is done by identifying names and types of variables, names of objects and methods, and replacing them with keywords with references to default values. Problems that should be resolved in this step are related to the language dependent syntax. Template generator must be able to generate templates which are not dependent on any language that consequently enables generation of production-ready source code in multiple programming languages using only one template.

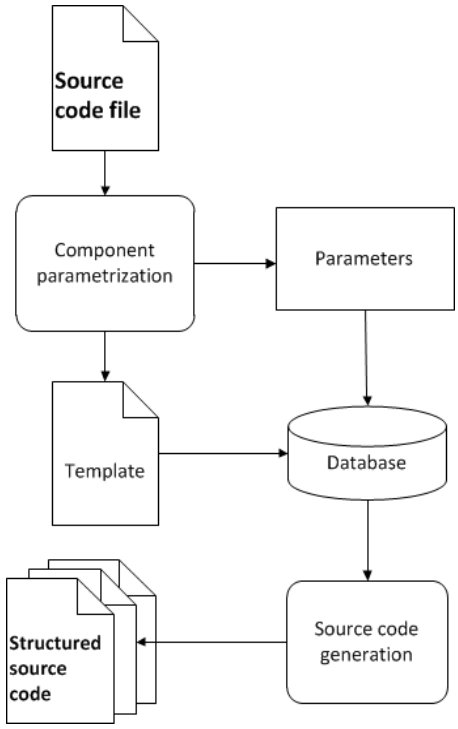

Fig. 4. Generation process - one source file

During that process, parameterized code would be turned into XSL template which could be saved in the database and would later (with suggested default values) be used as input to the source code generator. For code generation process, it is the best to use existing generator with ability to be adopted to own needs. In this case, open source generators, such as MyGeneration [15], are recommended choice.

B. The Advanced Model

The previous section describes the concept of a simple template generator that requires of developer to take a part of code, or one particular file that will serve as input to our program, and to put it in generator to be turned into template. This process speeds up the development of software and does not require any specific knowledge about template coding,

parameterization and metalanguages. However, developers work could be reduced even more as shown in Fig. 5.

The next step in the development of the concept of template generator is development of a model that would accept and analyze multiple files. Developer would, in that case, give as template generator input a number of files from the project. Those files should have a similar program code. The program would then, as in the initial concept, perform parameterization of source code. Parameterization would be followed by text mining process, which would also include algorithms for preservation of original source code of different class and definitions of particular programing languages.

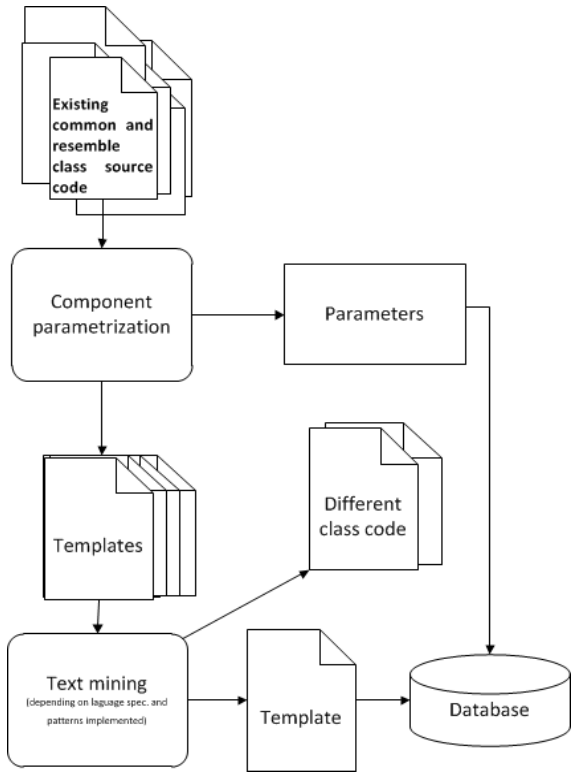

Fig. 5. Template generator model

The text mining process would take only parts of the code which can be converted into one or more templates, while other parts of code would be unchanged and their existence indicated to developer. Selected parts of code should then be turned into a template and saved in the database for further use. This model does not describe data migration - extraction of data from the existing system in order to re-format and re-structure it and to upload it into the new system, because mentioned processes are already well described in [16], [17], [18].

IV. Component Parameterization and Templates

The generator proposed in this paper combines multiple source code files into one or more templates. Multiple template files should be made when, for example, web application is subject of template generation [19], [20]. In that case separation of design and code, which should be supported by generators to produce structured and more manageable code, is welcomed. Proposed template language in this generator is XML and XSL. Parameterization and transformation of small code fragments should be supported as well as of whole application layers. Produced templates should be efficiently organized with browsing and searching options built into the user interface.

Each produced template can have zero or more parameters. To enable that the same template can be reused in several different applications or contexts, it must have zero parameters or it must have capability to generate source code based on user-defined parameters. If a template does not have parameters, its code is stored for later retrieval and in that case the generator acts as a code library producer and produces common source code class. If a template contains parameters than, during code generation, template parameters have to be populated with values provided by the user and inserted at specific places into the generated code.

A. *Simple Parameterization of the Source Code*

First process of template generator is to analyze source code, find variable/class names and types and store them into XML file as default values which will base values for new code generation. In new generation process those values will be provided to user as default, but user will be able to select other values depending on context of developing application. In order to use the existing code within a new context, it is necessary to define all above mentioned code fragments that could be changed in that context [11] as in C# example:

```
public class MyClass
{
    private int _age;

    public int Age
    {
          get{ return _age;  }
          set{ _age = value; }
    }
}
```

If we name, with unique labels, the code fragments that might be important to context, we get the following code:

```
public class [ClassName]
{
    private [VarType] _[VarName];

    public [VarType] [VarName1]
    {
          get{ return _[VarName];  }
          set{ _[VarName] = value; }
    }
}
```

The code used in example above represents a simple template with four independent parameters: ClassName VarName, VarName1 and VarType. As we intended, generated code could vary depending on the values entered for these four parameters. This simple template can be easily

created and implemented, but its syntax has at least two issues which prevent it from being used within a code generator:

- the syntax is language dependent (e.g.. if developer want to produce code in C++, this template will not be useful to him),
- there is no support for loops, conditions, comparisons and other statements.

To overcome the aforementioned limitations, it is proposed that the template generator uses XML and XSL as languages to describe templates as in [21]. XML is a language suitable for data description, while XSL is a powerful transformation language suitable to template description. Another advantage of using XML/XSL is that templates are stored in the way that they do not contain special syntax of some programming language, they are syntax free. During generation process, definitions of target language need to be loaded and production ready source code will be automatically generated in targeted language. Complex templates can be defined, and source code generated, using a combination of these two languages XML/XSL [22] as shown in Fig 6..

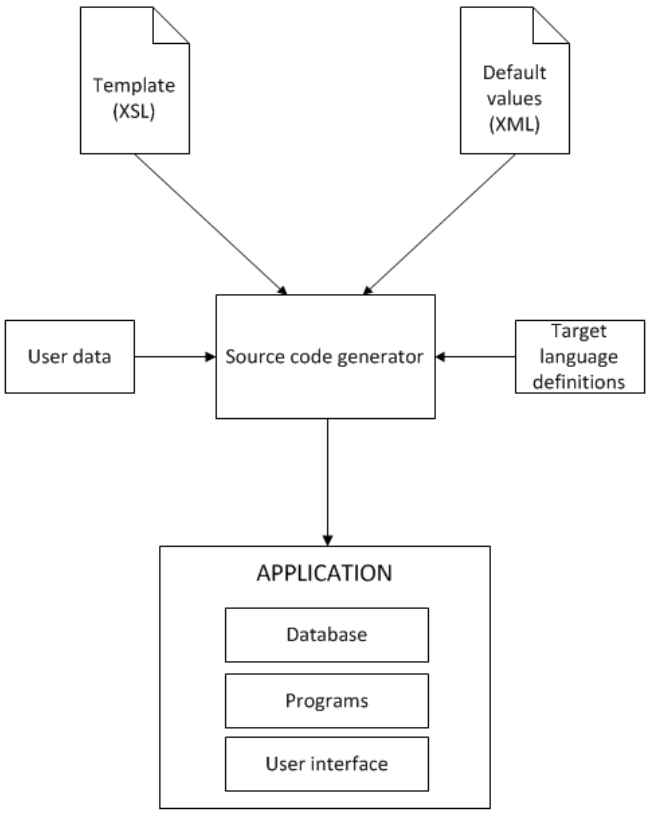

Fig. 6. Generation of application in target language

B. Template Parameters

Let us assume that proposed generator has to convert the following code into simple template. As previously mentioned, template will be first parameterized and then converted into XLS template, while data will be stored into XML format which will be shown in this section.

```
public class Person
{
  public int IDNumber {get; set;}
  public string FirstName {get; set;}
  public string LastName {get; set;}
  public DateTime DateOfBirth {get; set;}
}
```

If a template generator finds similar code describing same concern, in solution or files/classes given to analysis, it will try to make one template for code generation, but it will have to make multiple XML files for every occurrence of similar code. Those multiple XML files will be shown to the developer during source code generation, as list of default values.

Definitions of template parameters have to support dependency of one parameter on another. If template parameter file stores data about simple class without methods, it is obvious that for such class, during code generation, database table must be defined. In that case our data also depends on that database, table and table fields. During code generation, besides class code in a target language, developer must be able to generate and modify table in a database and the associated basic procedures:

- insert - inserts row in table,
- update - updates table row,
- delete (id) - deletes table row,
- select_one_row (id) – selects only one row depending on given primary key,
- select_all (order_by) – selects all table rows and orders them by given attribute name.

To enable mentioned requirements XML data template has to be improved by data related to database. Those parameters will be called *DatabaseTable* and *TableFields*. Last parameter, which indicates dependence on database, will not only contain database name but it'll contain also parameters to enable templates' connection to database. Last parameter will be called *ConnectionString*. Mentioned parameters are all mutually dependent and within XML dependencies must be defined that *TableFields* depends on *DatabaseTable* and it depends on *ConnectionString* [11]. Those parameter dependencies can be defined with hierarchically organization. Parameter values, organized on such way, allow easy refactoring of the XSL template. Furthermore, some fields must be allowed to have additional attributes like auto increment, data type, is NULL… Such attributes are needed because they enable generation of variable declaration and validation of generated code. The resulting format of XML in given example is:

```
<Param name="ConnectionString">
  <ParamValue value=”connection_string_value”>
    <Param name="DatabaseTable">
      <ParamValue value="Person">
        <Param name="TableField">
          <ParamValue value="CountryID"
            autoincrement="1"
            is_null="0"
            datatype="System.Int32" />
          <ParamValue value="FirstName"
            datatype="System.String" />
          <ParamValue value="LastName"
            datatype="System.String" />
```

```
          <ParamValue value="DateOfBirth"
            datatype="System.DateTime" />
        </Param>
      </ParamValue>
    </Param>
  </ParamValue>
</Param>
```

Usage of XML format is widely distributed among code generators as in [23].

C. Creating XSL Templates and Intermediate Code in Target Language

As stated previously, XML is used to describe data, but it is not powerful enough to be used to create templates. Instead of it, more powerful language is used XSL [24]. Let's take another simple example of interface containing methods for handling employees.

```
public interface IEmployee
{
  void OrderListByLastName();
  void OrderListDateOfBirth();
}
```

If this code is to be parameterized, first it is necessary to make XML data structure. Then, based on the source code and using XSL transformations, we get the XSL code [25]. Resulted XML data description, whose creation was described in the previous section, is:

```
<param name="Employee">
    <method name="OrderListByLastName"/>
    <method name="OrderListDateOfBirth"/>
</param >
```

After the XML code is created, the same source code is used to create XSL code. Tags in both codes, XML/XSL, should be equally for equal names and values. In XSL code, presented below, are two important characteristics:

- values, names of variables and method names are replaced with the *@name* which, later during production-ready code generation, must be replaced with specific default value from XML data. Through the graphical interface user would be able to change the default value in an arbitrary, if this value meets the rules of the target language,
- XSL has the ability to create loops (<xsl:for-each>). In that case there is no needed to define both methods from example through separate part of XSL code. It is possible to create loop and through it generate any number of methods, as shown in following code.

```
<xsl:template match="param">
public interface I<xsl:value-of select="@name"/>
{
<xsl:for-each select="method">
    void <xsl:value-of select="@name"/>();
</xsl:for-each>
}
</xsl:template>
```

As shown in previous example, generated XSL template does not contain the features of a particular language. It has only the features of object-oriented languages such as Java, C ++ and C#. This template can be used to generate production-ready interface code in any of mentioned languages. To demonstrate that ability, in example that follows, intermediate template code in C# and in C++ programming language is produced using XSL Code Generator with target language definitions [26]. Resulted template can afterwards easily be turned into source code in targeted language. Process scheme is shown at Fig. 6.

```
<!-- MAIN -->
<xsl:template match="/">
    <xsl:apply-templates select="param"
    mode="csharp"/>
    <xsl:apply-templates select="param" mode="cpp"/>
</xsl:template>

<!-- C# -->
<xsl:template match="param" mode="csharp">
  public interface I<xsl:value-of select="@name"/>
  {
    <xsl:for-each select="method">
      void <xsl:value-of select="@name"/>();
    </xsl:for-each>
  }
</xsl:template>

<!-- C++ -->
<xsl:template match="param" mode="cpp">
class I<xsl:value-of select="@name"/>
  {
  public:
  <xsl:for-each select="method">
      virtual void <xsl:value-of
      select="@name"/>() = 0;
  </xsl:for-each>
  };
</xsl:template>
```

At the end of the process, C++ code can be produced with normal code generators as [15], [23]. With this transformation we showed the way to translate normal source code into template and produce new source code as shown in Fig. 4. In order to make this demonstration more effective source code was written in C# language and target language for code generation was C++.

```
class IEmployee
{
public:
    virtual void OrderListByLastName () = 0;
    virtual void OrderListDateOfBirth () = 0;
};
```

As shown in this and previous section, it is not easy to create XML/XSL templates manually. It would be easier to choose other template language, since mentioned have a rather complex syntax. But XSL templates are much more powerful then alternatives, so further effort in the continuation of this research will be partly spent on creation of a tool with a user-friendly interface, which enables easy creation of templates. By using the tool, user will no longer need to learn the XML/XSL syntax.

D. Organization of Templates

Tool, mentioned in previous section, besides having user-friendly interface and enabling users to use it without knowledge of the XML/XSL syntax, must provide storage and organization of templates into categories in order to be easily managed. Also, it must provide search through large number of templates, it have to support keywords and enable users to associate their own keywords with each template. It is even possible to make automated but supervised classification technique for XML documents, which is based on structure only, and which would suggest pattern description as in [27].

V. Source Code Mining

After template metalanguage was described, this section will describe processes and algorithms of code mining. The objective of this code mining process is to determine similarities in code and suggest one resulting template which will be used to generate multiple files and classes. We will start with the assumption that certain similar problems during software development reoccurs. Those problems are solved by patterns. A pattern for software architecture describes a particular recurring design problem that arises in specific design contexts, and presents well-proven generic scheme for its solution [28], [29]. But, despite efficiency of proposed mining algorithms, some problems may occur [30]:

- old source code might be extremely complex or very poor quality,
- the amount of old source code can be rather large,
- the target model might still be subject to change.

Stated problems can, unfortunately, only be solved with developers' intensive labor and reverse engineering, process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction [31].

A. Crosscutting Concerns

In this paper we described potential advantages of template generators' use in practice. As said, it can be used in:

- continuation of incomplete existing software,
- software maintenance,
- development of new software versions,
- development of new modules of existing software,
- migration to new, developing platform,
- extraction and storage of templates with well-proven solution scheme for known problems to be used in other ongoing or upcoming projects.

Template generators could prove very helpful, but it will never be able to completely replace human labor and they will perform only one part of the job. Field on which the generators can prove their worth, and which also includes all of the above mentioned ways of generators' usage, are concerns - a parts of a software system that is relevant to a specific concept or purpose [32]. A key problem with software is that it is becoming larger and very complex, and much of this complexity can derive from the interaction of concerns [33]. An inadequate solution for crosscutting concerns implementation has a negative impact on the final system with consequences like duplicated code, scattering of concerns throughout the entire system and entangling of concern-specific code with code of other concerns. These consequences lead to software systems that are hard to maintain and to evolve [34]. Scattering of one concern through the entire system may be very harmful, as user logging. For example if it contains some kind of security vulnerability, and code is duplicated and scattered around, programmer who didn't participated in development of that particular code will hardly locate every piece of poor programming code.

Techniques for separation end encapsulation of concerns seek to cleanly disconnect concerns from source code in order to reduce complexity and increase comprehensibility [35], but a programmer faced with the task of identifying concerns in source code mainly has only intuition and experience to guide him or her. Also, challenge posed to programmers is to identify the full manifestation of specific concern in the entire source code.

B. Aspect Mining

Proposed template generator would aim to automatically discover above mentioned concerns and convert them into structured templates. It would be ideal for supporting aspect-oriented software paradigm because it focuses on the same problems as mentioned methodology: identification, specification and representation of cross-cutting concerns and their modularization into separate functional units as well as their automated composition into a working system [36], [37]. Template generator would undoubtedly enhance aspect-oriented software development and give it new boost by introducing it to existing lifecycle of software which was not developed by aspect oriented paradigm. Resulted templates could be used in new projects transferring encapsulation and separation of concerns from an old project to new one.

Core of template generator would be source code mining techniques. Those techniques must be able to identify scattered crosscutting concerns and turn each concern into well-structured encapsulated template so that produced code to can be easily understood, maintained and modified. Mentioned techniques are found in aspect mining, a relatively new research domain, but already with many aspects mining techniques have proposed as [38], [39], [40]. To be used in code generator, mentioned techniques should be refined and adopted to the generator. They must have ability to merge scattered concerns into one template, but also to remember default values of every instance of concern and provide them to the user via graphical interface so that average user does not need to know languages used to create template – XML and XLS.

VI. Conclusion

Maintaining the existing program code, or continuation of uncompleted software projects, can be very difficult and demanding job often done rather manually than with code generators because implementations of template engines are

most times based on practical experience rather than on a theoretical background [41]. Also many concerns in software may persist during its lifecycle even if application was developed with code generators. To facilitate the work of developers and help them to be able to focus on more important things such as development of new functionality and a new code, rather than time-consuming studying someone else's code, it's structuring and refinement, this paper proposes a model of template generator. Its purpose is, through aspect mining process, to find similar parts of code describing crosscutting concerns and turn them into one or more templates which will be stored for later development or code maintenance. Resulting templates could be used in other similar projects and would lean to aspect oriented paradigm.

Continuation of research activities on suggested model would include the development of prototype which, for the specific language and pattern database, could provide described results.

REFERENCES

[1] S. McConnell, *Rapid Development*, Microsoft Press, 1996.

[2] F. Alazemi, M. Alawairdhi, "Feature-based Approach to Bridge the Information Technology and Business Gap", Recent Researches in Telecommunications, Informatics, Electronics and Signal Processing, WSEAS Press, 2013, pp. 87-92.

[3] D. McDavid, "Systems engineering for the living enterprise", 18th International Conference on Systems Engineering, 2005, pp. 244 - 249.

[4] J. Herrington, *Code Generation in Action*, Manning Publications, 2003.

[5] R.B. Grady, *Successful software process improvement*, Prentice-Hall,1997.

[6] B.W. Boehm, *Software Cost Estimation with COCOMOll*, Prentice Hall, 2000.

[7] M. Yoshida, N. Iwane, "An Approach to the Software Product Line System for Web Applications", IEEE International Conference on Computing & Informatics, 2006, pp. 1-6.

[8] B.W. Boehm, *Software Cost Estimation with COCOMOll*, Prentice Hall, 2000.

[9] A. Cleve, N. Noughi, and J.-L. HainautI," Dynamic Program Analysis for Database Reverse Engineering", Generative and Transformational Techniques in Software Engineering IV: International Summer School, GTTSE 2011, Braga, Portugal, July 3-9, 2011, Springer, 2013, pp. 297-322.

[10] K. Fertalj, D. Kalpic, "Preservation of Manually Written Source Code in Case of Repeated Code Generation", Proceedings of the IASTED International Conference on Computer Science and Technology, 2003, pp. 38-43.

[11] T.Helman, K.Fertalj, "Application Generator Based on Parameterized Templates", Proceedings of the International Conference on Information Technology Interfaces - ITI 2004, pp. 151-157.

[12] M.C. Franky, J.A. Pavlich-Mariscal, "Improving Implementation of Code Generators: A Regular-Expression Approach", Proceedings of the XXXVIII Latin America Conference on Informatics (CLEI 2012), 2012, pp. 1-20.

[13] I. Liem, Y. Nugroho, "An Application Generator Framelet", Proceedings of the Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2008, pp. 794-799.

[14] K. Fertalj, D. Kalpić, V. Mornar, "Source Code Generator Based on a Proprietary Specification Language", Proceedings of the 35th Hawaii International Conference on System Sciences, 2002, pp. 283-291.

[15] M. Griffin, J.Greenwood, MyGeneration Code Generator, [Online]. Available: www.mygenerationsoftware.com

[16] K. Haller, *Data Migration Project Management and Standard Software: Experiences in Avaloq Implementation Projects*, Synergien Durch Integration Und Informationslogistik, DW 2008, St. Gallen, Switzerland. LNI, vol. 138. Gesellschaft für Informatik, 2008.

[17] K. Haller, "Towards the Industrialization of Data Migration: Concepts and Patterns for Standard Software Implementation Projects", CAiSE 2009. LNCS, vol. 5565, Springer, Heidelberg, 2009, pp. 63–78.

[18] J. Morris, *Practical Data Migration*. British Computer Society, 2006.

[19] J. D. Reilly, *Designing Microsoft ASP.NET Applications*. Microsoft Press; 2002.

[20] T. Helman, K. Fertalj, "A critique of web application generators", Proceedings of the 25th International Conference on Information Technology Interfaces - ITI 2003, pp. 639-644.

[21] M. Gangur, "The Use of XSLT for Table Data Tasks Generation", Proceedings of the 15th WSEAS international conference on Computers, 2011, pp. 503-508.

[22] T. Helman, K. Fertalj, "System for Automated Maintenance of Web Sites" Case studies of the Sixth European Conference on Software Maintenance and Reengineering, 2002, pp. 19-24.

[23] J. E. Smith, CodeSmith Template-based Code Generator [Online]. Available: http://www.ericjsmith.net/codesmith/

[24] J. Clark, The Extensible Stylesheet Language Family (XSL); W3 Consortium, [Online]. Available: http://www.w3.org/Style/XSL/

[25] J. Clark, The Extensible Stylesheet Language Family (XSL); W3 Consortium, [Online]. Available: http://www.w3.org/TR/xslt

[26] XSL Code Generator, [Online]. Available: http://www.codeproject.com/Articles/263100/XSL-Code-Generator

[27] P. Kumar, P. R. Krishna, S. B. Raju, *Pattern Discovery Using Sequence Data Mining:Applications and Studies*, Information Science Reference (an imprint of IGI Global), 2011.

[28] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal: *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns,* John Wiley &Sons, 1996.

[29] F. Buschmann, K. Henney, D. C. Schmidt, *Pattern-Oriented Software Architecture, Volume 5: On Patterns and Pattern Languages*, John Wiley &Sons, 1999.

[30] A. Rüping, "Transform! Patterns for Data Migration", Transactions on Pattern Languages of Programming III, Springer, 2013, pp. 1-16.

[31] E.J. Chikofsky, J.H. Cross, "Reverse engineering and design recovery: A taxonomy". IEEE Software 7, 1990, pp. 13–17.

[32] H. Ossher, P. Tarr, "Multi-dimensional separation of concerns and the hyperspace approach", Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development, 2000, pp. 293-301.

[33] M. Revelle, T. Broadbent, D. Coppit, "Understanding Concerns in Software: Insights Gained from Two Case Studies", Proceedings of the 13th International Workshop on Program Comprehension, 2005, pp. 23-32.

[34] G. Czibula, G. S. Cojocar, I. G. Czibula, "A Partitional Clustering Algorithm for Crosscutting Concerns Identification", Proceedings of the 8th WSEAS Int. Conference on Software Engineering, Parallel and Distributed Systems, 2009, pp. 111-116.

[35] G. Kiczales, J. Lamping, A. Mendhekar, C.Maeda, C. V. Lopes, J.-M. Loingtier, John Irwin, "Aspect-oriented programming", Proceedings of the European Conference on Object-Oriented Programming (ECOOP), 1997, pp. 220-245.

[36] A. Yakout, A. Mohamed, A. El Fatah, A. Hegazy, A. R. Dawood, "Aspect Oriented Software Development vs. other Techniques (Structured Approach and Object Oriented Approach)", Computer and Information Science Vol. 3, Canadian Center of Science and Education, 2010, pp. 256-276.

[37] A. Rüping, "Transform! Patterns for Data Migration", Transactions on Pattern Languages of Programming III, Springer, 2013, pp. 1-16.

[38] J.R. Avila, A.F. Ramirez, C.A. Cruz, I. Vasquez-Alvarez, "The Clustering Algorithm for Nonlinear System Identification", WSEAS Transactions on Computers, Issue 7, vol. 7, 2008, pp.1179–1188.

[39] G. Serban, G. S. Moldovan. "Aspect Mining using an Evolutionary Approach", WSEAS Transactions on Computers, 6(2), 2007, pp. 298–305.

[40] G. Moldovan and G. Serban, "Clustering based aspect mining formalized". WSEAS Transactions on Computers, 6(2), 2007, pp. 199–206.

[41] B.J. Arnoldus, M.G.J. van den Brand, A. Serebrenik, J.J. Brunekreef, *Code Generation with Templates*, Atlantis Press, 2012.