TECHNOLOGY ORIENTED REDOCUMENTATION OF LEGACY SOFTWARE

National Aviation University
E-mail: alena.avramenko@livenau.net

Abstract. The article describes the method of redocumentation of legacy software intended for using in software reengineering. Documents that are created in redocumentation should meet the requirements of software development technology to be used in forward engineering. Software document models located at different abstract levels are proposed: model of software document, meta-model of the document of the development technology, model of the development technology document. The models created are used in the method implementation based on the model-driven approach. According to this approach, redocumentation is considered to be the process of creating a set of legacy software documents by transformation models of documents, the documents contents being formed on the basis of legacy software views set.

Keywords: legacy software, model-driven approach, model of document, redocumentation.

Introduction

Redocumentation is the process of producing the documentation of existing and legacy software. Redocumentation originates from the increasing amount of legacy software and projects of reengineering and maintenance legacy software [1; 2].

In the 1980s such projects involved about 40% programmers, while in 2000 – already 60% [3].

The efficiency of reengineering projects implementation depends on detailed, accurate, and up-to-date software documentation. The documentation of legacy software is often obsolete, inaccurate, or doesn’t even exist. This requires redocumentation which is executed when stages of software development and software distribution have been completed and direct developers of this software are absent [2].

Related work

Redocumentation is a term that refers to a creation or revision of a semantically equivalent representation within the same relative abstraction level [1]. The use of reverse engineering for reconstructing the architectural aspects of software may be referred to as structural redocumentation [4]. As a result, the overall view of the subject system can be derived, and some of its architectural design information can be recaptured. In [5] redocumentation is defined as the process of analyzing the system to produce support documentation of various forms including users manuals and reformatting the system's source code listing.

Thus present day scientific researchers do not demonstrate how the results of redocumentation may be used in the reengineering process.

Purpose

This paper deals with the redocumentation method which is oriented to the software development technology and assigned for usage in legacy software reengineering.

Technology-oriented redocumentation method

The method suggested is intended for usage in reengineering of legacy software. Software reengineering includes two types of processes – reverse and forward engineering processes. Reverse engineering extracts information from legacy software. Forward engineering processes are the processes of the software development technology. The documentation created in redocumentation will be used in forward engineering. Therefore, the documentation shall meet the requirements of the software development technology. The essence of the method is that requirements to a legacy software documentation to be created shall meet the requirements of the software development technology applied in forward engineering and the contents of documents shall be composed of reverse engineering outcomes (fig. 1) [6].

“Depth” of performing legacy software reengineering depends on the stage at which forward engineering is started. Each forward engineering stage is based on the documents of the previous stage. Therefore redocumentation evolves creating the documents of only that stage which precedes the initial forward engineering stage.
Performing forward engineering in reengineering requires creating the documents of only one stage. These documents are a subset of the software development technology documentation.

Thus the redocumentation process in the method considered is controlled by the software development technology and the stage at which further software development should be started (fig.2).

**Models of a document in technology oriented redocumentation**

To develop the redocumentation method the MDD (model-driven development) approach is used. The essence of MDD as a software development methodology is in that software development may be represented by creation and transformation of coherent models describing the software at the different abstract levels: platform independent model, platform specific model and implementation model (program code) [7]. This approach allows to reuse the upper abstract level models and to automate their transformation if the models are produced using some formalized language. A document can be represented as a model that formally describes a document taking into account the general requirements to software documents as well as the requirements to redocumentation.

Requirements to redocumentation must be oriented to the requirements of the software development technologies to their documentation. The properties of the documentation of modern software development technologies (RUP [8], MSF [9], CDM [10]) may be applied to:

− documents of all technologies, such as the stage of software development at which a document is produced, the set of the identification elements (for example, version number, issue date) in a document. These properties have resulted from general requirements of all development technologies to the software documents;

− all documents of the particular technology, such as the set of general elements (for example, a table of contents, a glossary) in a document, the disposition of identification, general and basic (sections, subsections etc.) structural elements in a document. These properties have resulted from the requirements of the particular development technology to its software documents;

− specific document of the technology, such as the document structure, types of views which make up the content of a document. These properties have resulted from the requirements of specific development technology to specific software document.
The properties of the documentation of software development technologies underlie the software documents models which are placed at the different abstract levels:

- model $M_G$ of a software document at the level of a general concept of a software document. This model describes the general aspects of the document’s content and its structure. It takes into consideration the requirements to redocumentation rather than the requirements of the development technologies;

- meta-model $M_T$ of a document of development technologies at the level of development technologies. This model interprets the model $M_G$ by the refinement of the software document content and structure. It takes into consideration the general requirements $R$ of development technologies;

- model $M_D$ of a document of the development technology at the level of specific development technology. This model is the instance of the meta-model $M_T$.

-Model $M_D$ describes the content, structure and a form of specific document as metadata (meta-description). Meta-descriptions are used to generate the instances of the documents during redocumentation. Automation tools may be used for this purpose.

Model of the software document

Software standards define a software document as information designed for specific audience for some specific purpose using a specific medium of a particular format [11].

Software document model $M_G$ is presented by the triad $M_G = <S, C, P>$, where $S$ is the document structure, $C$ is the content and $P$ is the form of the document (appearance of the document).

Document structure $S$ is an ordered set $S = \{s_i \mid i = 1..M\}$, where $s_i$ is the structural element of the document (SED), which can be considered as the union $S = S_b \cup S_a \cup S_g$ of the three disjointed subsets of SED:
Information technologies

- basic SEDs ($S_b$), which are defined by the information content of the document (e.g. sections, subsections and paragraphs of the document);
- identification SEDs ($S_{id}$), which are designed to identify the document (e.g. document name and version number);
- identification SEDs ($S_{id}$), which are designed to identify the document (e.g. document name and version number);
- general SEDs ($S_g$), which are designed to search and navigate through the document (e.g. a glossary and a table of contents).

The set $S$ creates a tree $G = \langle S, E \rangle$, where $S$ is the set of nodes (SED); $E$ is the ordered set of edges, which present the SED hierarchy.

The basic and general SED can include the nested SED. The edges list defines the disposition of SED. For example, the edge list for the root SED $s_{id}$ is as follows:

\[
(s_{id}, s_1, s_2, s_3), (s_{id}, s_4, s_5, s_6), (s_{id}, s_7, s_8), (s_{id}, s_9, s_{10}, s_{11}),
\]

where $s_i - s_{id}$ are SED of the first (top) level (table of contents, chapters, annexes, index etc.).

Content $C$ of the document is the set $C = \{c_j \mid j = 1..L\}$, where $c_j$ is the informational element of document (IED). IED is a logical encapsulated element, which includes the information about software, e.g., class diagram, object diagram, interface specification. There is correspondence $O$ between elements of the sets $C$ and $S$. If certain IED $c \in C$ is defined, and then SED $s \in S$ is determined too. The correspondence is defined for any element of the set $C$. Hence, the correspondence $O$ is the mapping $O: C \rightarrow S$. IED is created by the inclusion of a view to it. A view is information about the software parts. The reverse engineering tools extract the views from the legacy (existing) software. The examples of the views are a requirements list, use-case or components diagrams, a class specification, a user interface screenshot, a DB scheme.

View $v$ has both value $z$ (text, figure, table, diagram, animation or sound) and properties $p_1, ..., p_n$ (type, notation, physical allocation of the value): $v = \{z, \{p_1, ..., p_n\}\}$.

During software redocumentation the views $v_i$ from set $V$ correspond to one or more elements $c_j$ from the set $C$. The correspondence of the sets $V$ and $C$ may be presented as matrix $A[K, L]$, where $K$ is the quantity of the elements of set $V$, $L$ is a quantity of the elements of set $C$. $A = \{a_{ij}\}$, $1 -$ signify that $v_i$ corresponds to $c_j$, $0 -$ vise versa. In addition to the view, IED may include the explanation $ex$ which has only text value: $z(ex)$, e.g., notes, title (footer) of tables and figures. The explanations create the set $EX = \{ex_i \mid i = 1, ..., M\}$.

Thus, for $\forall c_i \in C$ there the following can be determined:

\[
c_k :\Rightarrow \bigcup_{i=1}^{L} v_i^k \cup \bigcup_{j=0}^{M} ex_j^k,
\]

where $c_i \in C$, $v_i^k \in V$ and corresponds to $k$-th element of the set $C$, $ex_j^k \in EX$ and corresponds to $k$-th element of the set $C$.

The form of a document is defined by the style specification [11] and depends on document media.

**Meta-model of a document of the software development technologies**

Meta-model $M_r$ of a document of the software development technologies is created on the basis of model $M_g$ of the software document. It meets the requirements which are common for all development technologies $R = \bigcap_{i=1}^{N} R_i$, where $R_i$ are requirements of $i$-technology. When meta-model $M_r = \langle C(R^c), S(R^c), P(R^c) \rangle$ is transformed, it is parameterized by three parts of a document: content, structure, form of a document on the basis of the corresponding requirements $R^c, R^s, R^f$. The parameter of a document content is the notation of views $p_{notation}$. The parameter of a document structure is the part of graph $S$, which is defined for the set of identification SED $S_{id}$ and the set of general SED $S' = S_{id} \cup S_g, E'_{i}$. Where $E'_{i} \subseteq E$. The parameter of a form of a document is the set of formatting rules $F$. Rules are the common formatting and layout rules for documents of
software development technologies. Thus, parameterized meta-model looks like $M_T(p_{\text{rotation}}, S', F)$. The properties $\text{prop} = \langle p_{\text{tech}}, p_{\text{phase}}, p_{\text{name}} \rangle$ are added to the model $M_T$ by the requirements which are common for development technologies. These properties are connected with the development process: the name of technology $p_{\text{tech}}$ to which a document belongs, the name of stage $p_{\text{phase}}$ when a document is created and a document name $p_{\text{name}}$.

**Model of a document of the software development technology**

Model of a document of the software development technology $M_D$ is created on the basis of the model $M_T$. It meets the requirements $R$ of the software development technology to all documents. When the model $M_D = \langle C(R'), S(R'), P \rangle$ is transformed, it is parameterized by two parts of the document: content and structure on the basis of the corresponding requirements $R$ and $R'$. The parameters of the document content are the set of view types $p_{\text{type}}$ and explanations $EX$. The parameters of the document structures are the part of graph $S$ which are defined for basic SED $S_b$:

$$S'_2 = \langle S_b, E'_2 \rangle$$


Thus, taking into account the parameterization of the meta-model $M_T$, the model $M_D$ looks like:

$$M_D(p_{\text{rotation}}, p_{\text{type}}, A, EX, S, O, F).$$

**Realization of the technology-oriented redocumentation method**

The approach MDD is used to realize the technology-oriented redocumentation method. According to this approach redocumentation $RD$ is considered as the process creating a set of legacy software documents $D_{RD}$ by transformation models $M$ of documents with formation of the documents contents on the basis of the set of legacy software views $V$. Transformation of models is controlled by two parameters – chosen software development technology $T$ and stage $Ph$. Thus, $D_{RD} = RD(M, V, T, Ph)$. Realization of redocumentation founded on MDD is the execution of the following transformations (fig. 3):

- model of a software document $M_G$ to a meta-model of the document of the development technologies $M_T$ through refinement taking into account the requirements $R$, which are common to all software development technologies;
- meta-model $M_T$ to a model of the development technology document $M_D$ through refinement taking into account the requirements $R_i$ of the chosen software development technology;
- model $M_D$ to a meta-descriptions $m_D$ of documents of the chosen software development technology through of substitution in parameters of model $M_D$ the actual values, which are defined by the requirements $R_{ij}$ to $j$-document.

The last step of redocumentation realization is the creation of legacy software documents $D_{RD}$ as instances of meta-descriptions $m_D$.

For documents within one software development technology there is only one transformation of models. Documents are created on the basis of prepared meta-descriptions using the processes of translation and integration, which can be automated. In order to automate the suggested redocumentation method, CARSE facilities architecture is proposed [12]. Facilities execute the followings functions: preparation of meta-descriptions of documents, preparation of legacy software views and creation of documents.

**Conclusions**

Application of model-driven approach to technology-oriented redocumentation allows formalizing and automating the document creation processes. The advantage is to reuse models in different reengineering projects which are oriented to the same software technology and stage of forward engineering. The advantage is to simplify the creation of documents with the same content and structure but of different forms because they are created on the basis of the same models.
Fig. 3. Creation of legacy software documents $D_{RD}$ in technology-oriented redocumentation

References


2. Сидоров, Н.А.; Хоменко, В.А.; Авраменко, Е.А. Реконструкция проектов программного обеспечения // Проблеми програмування. – 2006. – № 2.3. – C. 31–38.


Received 10 March 2011.