Conceptual model of an application and its use for application documentation

Martin Vonka
Faculty of Civil Engineering
Czech Technical University in Prague
vonkama@gmail.com

Abstract: Following article proposes methodology for conceptual design of a software application. This form of design is suitable for dynamic development environment and agile principles of software development. Article discusses the required scope and style used for description of the application. Unification of a documentation significantly reduces the time required for communication within the development team. Some part of the documentation are obtained using the method of reverse engineering, for example by analysis of the application structure or its source code.

Key words: software development, documentation, application concept

1. Introduction

This article deals with the design methodology on how to build a conceptual model of an application. Using this model in the development of the application itself and during the preparation of a documentation required for the final product. Therefore, not just the operational and maintenance documentation, but also the user documentation. Using the principles of application design described in this article it is possible to achieve significant time savings during the actual development as well as during documentation process. This should be the primary objective for every developer - effectiveness in the entire application lifecycle. Software development is undergoing in recent years a great innovation. There is a lot of new software development companies applying agile methodologies of software development. Clearly defined developments phases are fading out. - taken from [Weske, 2007]:

- requirements definition
- design and project documentation
- programming
- testing
- customer hand off, documentation updates
- maintenance and operations
- decommissioning

These tightly defined phases are being replaced by dynamic development methods based on the principles of extreme programming and the spiral model.

This article deals with basic principles that are necessary to build a conceptual model for the application development [Wikipedia 2] and the subsequent management of the entire software project. So the final outputs of the development process will be an actual application, administrative manual and user documentation. At the same time, this methodology takes into account the requirements and possibilities of modern development methods that can be used in a dynamic development environment.

The goal is to automatically assemble the documents, or document parts from the current state of the application. These documents can be divided into three categories, user documentation, documentation for developers and documentation for the administrator.

User documentation consists of the following parts:

- User documentation, or components for its compilation
- Documentation for individual parts of an information system using following breakdown:
  - Documentation for application = general description of the features of the application
Conceptual Model of an Application and Its Use for Application Documentation

- Documentation for data representation (tables, graphs) that information system provides
- Documentation for forms and its purpose, expected values, the precise interpretation of values ...

- Description of the implemented processes in the form of diagrams

Developer Documentation consists of the following parts [Makowski, 2010].

- Description of the application structure
- ER diagram
- Developer Documentation
- Description of functions including a description of the parameters and the methods of key values calculations.
- Analysis and control of an application
- Proper index settings
- Application function calls
- Authentication, to prevent attempts to call unreachable values

Administrator (operational documentation) - used for solving problems at runtime

- SQL queries execution time analysis
- Loading time of forms and tables presenting data to the user
- Execution time of multiple queries (join) and the use of joint columns

2. Solving the issue of documentation by reverse engineering

Software can be seen from different angles. The user sees the software or information system as a set of processes and functions [Řepa, 2006], which is used within the organization. The user often has no idea about architecture or method of functioning of the software. The important parts for the user are inputs and outputs of the functions that he/she is using.

The definition of reverse engineering by Elliot Chikofsky: "Reverse engineering is an analytical process of the system, which leads to the discovery of the system components and their interconnections, followed by description of the system in another way or by using a higher level of abstraction." [Eilam, 2005]. Methods of reverse engineering, approach the system as a whole and try to decompose it into subsystems. Relationships between system components are simplified and recorded using the available methodologies.

A prerequisite for a successful decomposition of the system is to understand how it is arranged within. [Řepa, 2006]

For software programmers, the entire information system is defined by its components (subsystems), architecture and data flow between components. On the lower level (single component) is represents a specific set of software libraries, classes and their source codes which process data. In addition to individual components, then the programmer identifies each data source and method of communication with them.[PMI, 2004] Software structure could be divided into following layers according to their perception:

- Complex functions provided by information system to the user (inputs, outputs)
- The system architecture - components and data flows
- Feature specific components
- Functions source code
- Data – methods of data processing in an information system

If the operator of the information system has access to all these layers and has control over it, then he is able to modify and develop the system going forward. Otherwise operator uses information system functions and has no control over data processing. In order to gain this control it is necessary to examine the system to map the structure and functionality of the individual layers and data flow. This analysis is called reversing, or reverse engineering - by [Aalst, 2004].
2.1 Requirements definition

The goal of reverse engineering is to identify system components and their relationships, the initial steps are:
- create a representation of a system in another form (at a higher level of abstraction) - therefore any reality describing model
- inspection of the system – verification of links and bug finding
- preparing structures for further processing of data (data mining)
- documentation updates

2.2 Sources of study

For the study of "finished" applications we have following sources of information:
- source code (the systematic “top-down” study)
- architecture of the application
- database structure
- original documentation, which may be misleading in many aspects due to changes in project implementation

2.3 Tools for generating documentation from source code

Here I present list of available programs that are able to generate documentation from a properly commented application source code. In some special cases it is possible to generate at least the structure from objects and the already compiled executable file – or from dll library. These libraries are for example .NET binary executable files, which contains that .NET framework is using during the execution.

Next table is taken from [Wikipedia 1] and [Kerzner, 2003]. It contains a brief overview of programs designed for maintenance of project source code and documentation. This list contains only tools that can work with source code.

<table>
<thead>
<tr>
<th>Name</th>
<th>Creator</th>
<th>First public release date</th>
<th>Latest stable version</th>
<th>Software license</th>
<th>Text</th>
<th>Bin.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApiGen</td>
<td>David Grudl, Jaroslav Hanslík, Ondřej Nošpor</td>
<td>2010</td>
<td>2.5.0</td>
<td>Now BSD / GPL v2 / GPL v3</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Appledoc</td>
<td>Gentle Bytes</td>
<td>2009</td>
<td>2.0.3 [1]</td>
<td>Modified BSD</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>BI Documenter</td>
<td>Mienasoft</td>
<td>2006</td>
<td>1.5</td>
<td>Proprietary</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>cclassdoc</td>
<td>Jens Gulden</td>
<td>2001</td>
<td>1.0</td>
<td>GPL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Codenotes (format convention; not a generator)</td>
<td>Now Vintage Media</td>
<td>2012</td>
<td>1.0</td>
<td>Omnilitensue</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>CppDoc</td>
<td>Richard Feit</td>
<td>1998/11/23</td>
<td>2.3.1</td>
<td>Proprietary</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>DB Manual</td>
<td>Blue Oceans Ltd</td>
<td>2006</td>
<td>1.3</td>
<td>Proprietary</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>DBDesc</td>
<td>Logica2</td>
<td>2005</td>
<td>3.1</td>
<td>Proprietary</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DBScribe</td>
<td>Leadum Software</td>
<td>2007</td>
<td>1.1</td>
<td>Proprietary</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Ddoc</td>
<td>Walter Bright</td>
<td>2005/09/19</td>
<td>DMD 1.010 / GDC 0.14</td>
<td>Freeware / GPL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Name</td>
<td>Creator</td>
<td>First public release date</td>
<td>Latest stable version</td>
<td>Software license</td>
<td>Text</td>
<td>Bin.</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------</td>
<td>---------------------------</td>
<td>-----------------------</td>
<td>----------------------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>devsccribe</td>
<td>Geoff Cox</td>
<td>2008</td>
<td>2.00 [4]</td>
<td>MSDN Code Gallery Licenses</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DOC++</td>
<td>Dragos Acostachioale.</td>
<td>2002/12/22</td>
<td>3.4.10</td>
<td>GPL</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Doc-O-Matic</td>
<td>toolsfactory software inc</td>
<td>2000/11/--</td>
<td>6.1</td>
<td>Proprietary</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Document! X</td>
<td>Innovasys</td>
<td>1998</td>
<td>2014.1</td>
<td>Proprietary</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Doxygen</td>
<td>Dimitri van Heesch</td>
<td>1997/10/26</td>
<td>1.8.7</td>
<td>GPL</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Epydoc</td>
<td>Edward Loper</td>
<td>2002/01/--</td>
<td>3.0</td>
<td>MIT License</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ForgeDoc</td>
<td>Balazs Tihanyi</td>
<td>2012/02/21</td>
<td>1.0</td>
<td>Proprietary</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>fpdoc</td>
<td>Sebastian Guenther</td>
<td>2005?</td>
<td>2.4.0</td>
<td>GPL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>GenHelp</td>
<td>FraserSoft</td>
<td>1998/10/--</td>
<td>5.6.7</td>
<td>Proprietary</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Haddock</td>
<td>Simon Marlow</td>
<td>2002</td>
<td>2.13.1 (2012)</td>
<td>BSD</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>HeaderDoc</td>
<td>Apple Inc.</td>
<td>2000/09/--</td>
<td>8.9.14</td>
<td>APSL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Help GeNorator</td>
<td>Agora Software BV</td>
<td>2004/08/--</td>
<td>4.0</td>
<td>Proprietary</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>HyperSQL</td>
<td>Randy Phillips, Itzchak Rehberg</td>
<td>2001</td>
<td>3.7.5</td>
<td>GPL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IDLdoc</td>
<td>Michael Galloy</td>
<td>2002</td>
<td>3.3.1</td>
<td>BSD</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Imagix 4D</td>
<td>Imagix Corp.</td>
<td>1995</td>
<td>7.3</td>
<td>Proprietary</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Javadoc</td>
<td>Sun Microsystems</td>
<td>1995</td>
<td>1.6</td>
<td>GPL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>jGrouseDoc</td>
<td>Denis Riabtchik</td>
<td>2007/04/--</td>
<td>1.5</td>
<td>BSD</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>JSDoc</td>
<td>Michael Mathews</td>
<td>2001/07/--</td>
<td>1.10.2</td>
<td>GPL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>JsDoc Toolkit</td>
<td>Michael Mathews</td>
<td>2007?</td>
<td>2.0.0</td>
<td>MIT License</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>KDOC</td>
<td>Sirtaj Singh Kang</td>
<td>1999</td>
<td>3.0.1</td>
<td>?</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>mdk</td>
<td>Jean-Paul Louyot</td>
<td>1989</td>
<td>12.03 (2012)</td>
<td>EUPL</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Name</td>
<td>Creator</td>
<td>First public release date</td>
<td>Latest stable version</td>
<td>Software license</td>
<td>Text</td>
<td>Bin.</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------</td>
<td>---------------------------</td>
<td>-----------------------</td>
<td>------------------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>Natural Docs</td>
<td>Greg Valure</td>
<td>2003/05/26</td>
<td>1.52</td>
<td>GPL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>NDoc</td>
<td>Jason Diamond, Jean-Claude Manoli, Kral Ferch</td>
<td>2003/07/27</td>
<td>1.3.1</td>
<td>GPL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Pandoc</td>
<td>John MacFarIaNo</td>
<td>2006</td>
<td>1.11.1</td>
<td>GPL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>perlDoc</td>
<td>Larry Wall</td>
<td>1994</td>
<td>5.16.3</td>
<td>Artistic, GPL</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>phpDocumentor</td>
<td>Joshua Eichorn</td>
<td>2000</td>
<td>1.4.4</td>
<td>LGPL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>phpSimpleDoc</td>
<td>Thierry Graff</td>
<td>2008</td>
<td>1.0</td>
<td>GPL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>PLDoc</td>
<td>Albert Tumanov</td>
<td>2001</td>
<td>0.9.1</td>
<td>LGPL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Project Analyzer</td>
<td>Aivosto Oy</td>
<td>1997?</td>
<td>9.0</td>
<td>Proprietary</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>RDoc</td>
<td>Dave Thomas</td>
<td>2001/12/14</td>
<td>in Ruby core</td>
<td>Ruby License</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ROBODoc</td>
<td>Frans Slothouber</td>
<td>1995/01/19</td>
<td>4.99.36</td>
<td>GPL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Sandcastle</td>
<td>Microsoft</td>
<td>2008/05/--</td>
<td>2014.5.31.0</td>
<td>Ms-PL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SoDa</td>
<td>IBM</td>
<td>2008/07/--</td>
<td>7.0</td>
<td>Proprietary</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Sphinx</td>
<td>Georg Brandl</td>
<td>2008/03/21</td>
<td>1.2.2</td>
<td>BSD</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SQL Documentor</td>
<td>InfoCaptor</td>
<td>2005/11/--</td>
<td>3.4</td>
<td>Proprietary</td>
<td>Text</td>
<td>Binary</td>
</tr>
<tr>
<td>ThimbleDoc</td>
<td>Thimbleopensource. com</td>
<td>2008</td>
<td>0.2beta</td>
<td>GPL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>TwinText</td>
<td>PTLogica</td>
<td>2002</td>
<td>2008.0.406</td>
<td>Proprietary</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Universal Report</td>
<td>Universal Software</td>
<td>2001–2010</td>
<td>9.0</td>
<td>Free Education / Proprietary</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>VBDOX</td>
<td>Mihayl Stamenov</td>
<td>2001/10/26</td>
<td>2.1.0</td>
<td>GPL</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>VBSdoc</td>
<td>Ansgar Wiechers</td>
<td>2010/11/29</td>
<td>2.3</td>
<td>GPL</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>VSdocman</td>
<td>Helixoft</td>
<td>2000/10/--</td>
<td>6.8</td>
<td>Proprietary</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>WEB/CWEB</td>
<td>Donald E. Knuth</td>
<td></td>
<td></td>
<td></td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>YARD</td>
<td>Loren Segal</td>
<td>2007/02/24</td>
<td>0.7.3</td>
<td>MIT License</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### 2.3.1 Inputs for an analysis

A very important parameter for evaluation of individual SW is the format of input data which will be processed. The last two columns of the table above indicates if the software allows the analysis of source codes - column "Text". Or if it allows the analysis of already compiled programs - column "Bin." In cases where the analysis is used during programing, source codes are available and it's possible to work with them. These types of analyzes are focused to support the team of developers. They keep the information about the structure of classes, object properties or manage documentation and description of functions written in the comments placed directly in the source code.
A second approach of the analysis - the analysis of the compiled binary files, is performed when for some reason we do not have access to the source code and we don’t know its structure and how it is programmed.

2.3.2 Supported Environments

The above listed programs for managing source code and supporting work on software development teams, works with almost all programming languages. Here is the list of the common one: C/C++, Java, C#, VB / VBScript, Delphi / Pascal, Ada, D, IDL, .NOT1, Access, Fortran, PHP, Perl, Python, Ruby, JavaScript, ActionScript, PL/SQL, Tcl, Haskell, AppleScript, MIG, BourNo Shell, C shell.

2.3.3 Output format

The outputs of individual programs are always in HTML format, eventually in other formats such as CHM, RTF, PDF, LaTeX, PostScript, man pages, DocBook, XML, ePub.

A more detailed description and capabilities of the individual programs are described in [Wikipedia 1].

2.4 Generating the structure of database

Another part of the documentation is ER Diagram of the database. It is possible in some cases to generate ER from an existing database. This functionality is only available for certain database servers and only under certain circumstances. It is important that the tables have been linked at the level of the database. Setting up links in the database server can in some cases replace the identification of indexing columns that are used for bindings, or at least suggests to SQL Server what data it should include in the database cache. [Kerzner, 2003]

Relations between the tables, may also be maintained at the application level. In this case, database reverse engineering is more difficult, in some cases impossible. Tools for generating diagrams from DB structures usually expect relations to be defined on the database level. In this chapter is a brief overview of the tools which enable to generate this type of documentation. There is also description of conditions that must be met in order to be able to generate ER Diagram from database. The following paragraphs describe some tools that allow automated generating.

3. Design of methodology for creating conceptual model of an application

For application development is good to use a modern flexible development methods and tools. When using today’s dynamic development tools it is a good practice to manage the development project using the spiral methodology and extreme programming practices.

It’s desirable to respect the mentioned methods when creating initial documentation - specification. The specification may not accurately describe all the details of applications at the lowest level. The level of simplification of application design must be chosen very carefully so that the basic model of application (either procedural or structural) contains and identifies all the parts (components) of the applications, all data flows and all relevant processes. The specification must also have clearly defined objectives of the project, according to which the application will be gradually developed and subsequently accepted. [Richards, 2007]

The initial proposal is therefore used mainly for relatively accurate estimate of the complexity of application development - work involved most often calculated in-man-days and subsequently in money. Such a proposal is then attached as part of the contracts under which is the application being developed.

In other iterations of the project, the individual parts of the proposal are sequentially or in parallel (parallel - if it their character allows) described in more details to create specific tasks for development team.

During the detailing of the specification it is a good practice to think about it as documentation for the whole application, so that the individual parts can be integrated into the final product documentation without major modifications.

In iterative development of an application it can come to the changes to the fundamental design of the original logic of the entire applications. The amount of major changes depends on quality of the original conceptual design, eventually on the experience of the managing person of each iteration. It is important to keep the iterations within the contracted budget and timelines.
3.1 Structure of the conceptual model

Conceptual model should contain the following fundamental parts:

- **Goals**
  - the essential functionality provided by the application
- **Object schema**
  - the description and the links between different parts (components) of an application
  - Communication with other systems - data exchange
- **Basic ER Diagram**
- **Description of tables and their relationships**
- **Description of the processes that will be implemented in the program**
  - the list of functions that will be used by the process
- **Acceptance criteria and methods of verification**

3.2 Setting goals

Setting goals is based on the specification of the information system. It describes what inputs and outputs the program will have, what functions it will provide and what processes it will support.

**Form**

The form of the document is mostly text with diagrams illustrating the basic structure of the application or simplified processes.

3.3 Object schema of an application

Object schema complements a text description of the goals of targets of an application. It should provide a clear graphical representation of the basic components of the application and the underlying data flows. The emphasis is on clarity.

**Form**

Any graphical representation of the main processes and the structure of an application.

**Example**

An example of an application is a diagram solutions that provides data from maps. Data are presented as web and mobile application.
3.4 Basic ER Diagram

The ER diagram should contain fundamental tables on which the application is built. The diagram must include all relevant relations and a relations table. Diagram is used as the initial design. During the each iteration of development it may be partially modified or supplemented with other functions.

Diagram of a final design of an application, to reflect database structure changes during each iteration of development, it is no longer appropriate to incrementally update the diagram, but directly generate it from the final application. The current state of the application is reflected in the documentation - correct ER diagram.
3.5 Description of the basic processes that will be implemented in the program

The process description must include all the important decisions and states. Flowchart defines application functions being implemented that validate the various inputs and allow the user to continue the process further, or eventually stop it and request additional information. Automatic functions if they exist in the application are programmed based on these processes.

Form

Description of the process usually consists of two parts. The graphical representation – i.e. the flowchart and text description of the each function. For graphical representation as well as for a text description the "less is more". Thus, there is an emphasis on proper structuring of the text and simplicity of the whole process scheme.

Example

In the example is shown a graphical diagram of the process and part of a textual description of the functions used in the process.
Example text description of the functions of the above process diagram:

1: 
**input:**
The user enters into the mobile phone information about fuel prices on the selected fuel station sends data it to a specific phone number.

**action:**
3rd party application receives the request.

**output:**
None

2: 
**input:**
Timer initiates event every 2 minutes

**action:**
The application connects via SMS API and downloads all new data from SMS center.

**output:**
Raw SMS message in the application NET Genium.
- syntactically correct SMS

Example description of the process of the project for prices processing

---

**Figure 3: Graphical representation of the process - flowchart (own work)**
3.6 Acceptance criteria and methodology for verification

Acceptance criteria are a very important part of the conceptual design, they offer a checklist of the required functionality and how the individual functions or support processes will be controlled during the acceptance phase of application deployment.

4. Time schedule and budget

When the conceptual design of the application is created and all goals are written, it is necessary to estimate the time required for the development of individual components and plan budget for the entire project implementation. In most cases the conceptual design of the application should be sufficient for budgeting and identification of other costs that may occur during project or other project expanses.

Examples of additional costs are licenses for operating systems or database, as well as other supporting software necessary.

For the purpose of the cost and time estimate it is good to divide the project into separate units, which will be subsequently assigned to individual developers. Such a structured estimate, is possible to use as assignment of tasks during the project realization.

For a project in the value of 100 hours is the estimated amount of work involved compiling the documentation about 10%, therefore 10 hours. If we use a new methodology of simplified application design then we can generate documentation with final formatting within 2 hours. It is necessary to also add time, which is spent by developers to enter the documentation into the application. In total we are looking at seven hours of work, comparing to the standard method. This represents about 30% less work involved in the preparation of application documentation. However, another area of a substantial savings is the application development itself. Developers are not forced to study the extensive project documentation. It is sufficient for them to get familiarity with the concept of an application. When solving individual tasks they use documentation embedded directly in the application and are only concerned with the parts that are directly related to their task.

5. Development resources planning

When using the spiral method of software development, it is necessary to carefully plan of each development iteration (design - development - testing - assessment - proposal2 - ...) and stick to the milestones established for the project and budget.

For the entire project is necessary to properly set the individual phases, which will correspond with individual part or components of developed software. Each iteration runs within these phases, of the spiral model.

For each development iteration in the given phase is necessary to compiled basic structures of specific tasks. In the analytical phase, requirements are being defined in details.

For each part of the application is necessary to prepare:
- Database structure and forms that will be used for data entry.
- In parallel, create functions that will be used to handle the forms (transformation and calculations with data for one record) and processing of database records (multiple records transformation, verification of states, privileges ...)
- Setting permissions, data access, creation of the data on which the implemented functions and process are verified.
- The plan for testing of the application. It is very important to document all possible combinations of inputs and resulting outputs. Furthermore, the test scenarios for all possible variants of the user steps through the flowchart of implemented processes.

6. Conclusion

The methodology described above for compiling the concept of an application is designed primarily for dynamic development environments. These environments allow you to minimize the time which is necessary for maintaining project documentation. Documentation in these environments are provided within the application itself. During the development of the individual parts of the application, based on models and principles described in this paper, are directly updated and further developed. The
information contained in the application during its development are then used for the preparation of final documentation. For the meaningful use of assembled documents in practice, it is very important to follow the rules described above. From a personal experience I can confirm that the time invested in training of individual developers and continuous control of the partial development results, contributes to the significant time savings during the management of documents that must be submitted along with the completed application before acceptance.

Another advantage of this approach is the unification of documents for developers. Especially when a new developer starts to work on the project. Then the developer can easily find all the documents for the analysis of the problem with the application or during the implementation of a new functionality. This systematic approach, according to my estimates, will save approximately 30% of the time required to maintain the documentation.

**Literature**


**JEL Classification:** L86, M15