# A METHODOLOGY FOR SOFTWARE DOCUMENTATION

Roberto Dias Torres Júnior Prof. Dr. Hubert Ahlert

Universidade Federal do Rio Grande do Sul

Centro de Processamento de Dados – Divisão de Sistemas de Informação

[rtorres, hubert] @cpd.ufrgs.br

## ABSTRACT

With the growing complexity of window based software and the use of object-oriented, the development of software is getting more complex than ever. Based on that, this article intends to present a methodology for software documentation and to analyze our experience and how this methodology can aid the software maintenance.

Key words : Software Engineering, Methodology, maintenance, documentation, walkthroughs.

## 1. Introduction

The way that business takes place has changed in the last 50 years. With the desktop computers and the Internet, the relationship among people became very different [REZ 97].
The creation of databases allows an extremely fast access to information, never imagined before. But associated to this, the complexity of software is increasing and because of this, software maintenance is increasing, too.
Documentation is very important to aid software maintenance and this paper intends to analyze its advantages, how it can be applied and maintained.
For that, a methodology and a case study will be presented to analyze the results.

Ps.: It´s important to freeze that this paper will cover just the window and code documentation, and not all complete software development phases.

## 2. Software Documentation and Maintenance

Software maintenance is the phase that involves all changes after the user has received the software, going on until the software is dead [PRE 87]. One of the concerns of software staffs today is how to maintain the existing portfolio of programs.
Consider the following maintenance problems [REZ 97]:

- Most computer programs are difficult and expensive to maintain
- Software changes are poorly designed and implemented
- The repair and enhancement of software often injects new bugs that must be later repaired

Because of those and other problems that will be shown, the software maintenance process requires special techniques. It is desirable that all company has a methodology aiming for effectiveness, continuity, safety and transparency [REZ 97]. Managers, users and developers of software must accept this methodology.
The justification can be formalized so that your products can [REZ 97]:

- supply the project status at any instant
- detail the levels adapted to the administrators' interests
- serve as a tool of communication among staffs
- indicate the participation level of staffs
- keep any documentation about the software
- serve as basis to the upcoming steps

Let’s see some data about software maintenance [PAR 86]:

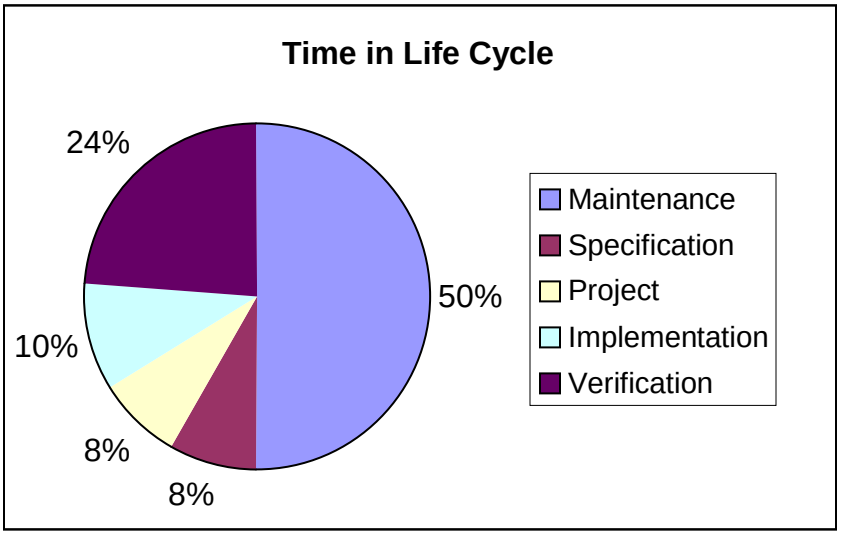


Figure 1. Time in Life Cycle

These data are statistical, but commonly they correspond to the truth.

## 3. Motivation and Objectives

The main reason that brought to the development of this methodology is that all systems produced on CPD- UFRGS ("Centro de Processamento de Dados") are being very different and hard to maintain.
In 1992, we start changing the architecture from mainframe to client-server. On that platform we are using Sybase Database Management System, with visual languages to develop the systems. Nowadays, each programmer makes his system without any documentation, except comments in the program source. That happened because two or three years ago there were only two programmers, turning the maintenance simpler. But now there are more than five, and the communication among programmer's is getting harder. This methodology pretends to establish some patterns to be followed by all systems (to exemplify the patterns, we will adopt the Delphi language [LEA 98]).
This methodology is based on the fact that all programs have source code, and it must be commented. For that, some patterns about code documentation will be presented. Besides, as all systems have many routines (methods), we are presenting some techniques to document them with skeleton of routines. Finally, we'll show a pattern to aid software maintenance developed with visual languages.
Besides the advantages previously mentioned, let's enumerate others obtained using a pattern in software documentation [REZ 97].

- It allows to detail, in the appropriate levels, the interests of the team
- It makes the communication among staffs easier
- It forms documentation of the system
- It makes then maintenance easier
- It serves as basis for the following phases
- It increases the development speed of a system

## 4. Patterns

Let's describe now the established patterns. The documents that are not in code need to be printed in a kind of commercial paper, as shown below:


Name of Company
Name of Department
**Name of the System**

### 4.1.Window description

It should have a complete description, between comments, in the beginning of a code for each window, explaining what it does. The name of the programmer and the last change are mandatory.

Ex.: { ----------
This window is responsible for the *enrollment* of the students. To do so, access the table STUDENTS, using the component QueryStudents.

Programmer: programmer's name Last change : 31/03/2000
---------- }

*Benefits*: it allows having a partial or a complete idea of the functionality of the window, not needing to analyze the code.

## 4.2.Methods division

All methods need a split with line comments, with the name of the method between lines.
Ex.:

```
{ ************************************************************
                          CREATE
  ************************************************************ }
     code for create

{ ------------------------------------------------------------------------------
                          MY METHOD
  ------------------------------------------------------------------------------ }
     code for my method
```

*Benefits*: the code is cleaner and clearer, turning easier the search of a routine inside the code.

## 4.3.Window debugging

It consists in a way to allow the analysis of all common requirements of the windows. If all items are right, an X should be marked in the "result" column, and at the end of the document, the responsible programmer needs to fill out the data and sign the document.
Ex.:

| Test | Result |
|---|---|
| Fields only accept valid data | |
| TabOrder right | |
| Data are being recorded correctly in the Database | |
| If there is no data in the database, any problem? | |
| Are comments in agreement with standards? | |

*Benefits:* it makes easier to check if everything in a window is perfectly working, allowing to remember commonly forgotten details and also allows an effective control, because the documents were signed.

## 4.4.Guidelines for window construction

Each system needs specific guidelines, being a document that describes, step-by-step, the phases needed to make a window with the basics functionalities.
Ex.: Create Event

- Configure the height and width
- Configure DataSetAtivo
- Call AlteraEstadoFormulario method

Activate

- Configure TitleBar
- Configure Button status

*Benefits:* Besides accelerating the construction of a window, it allows to remind of small details.

## 4.5.Abstract Window Description

It is a document where the hierarchical level of a window, a picture and the main methods and events are shown. Besides this, the visual components used in the window are listed.
Ex.:

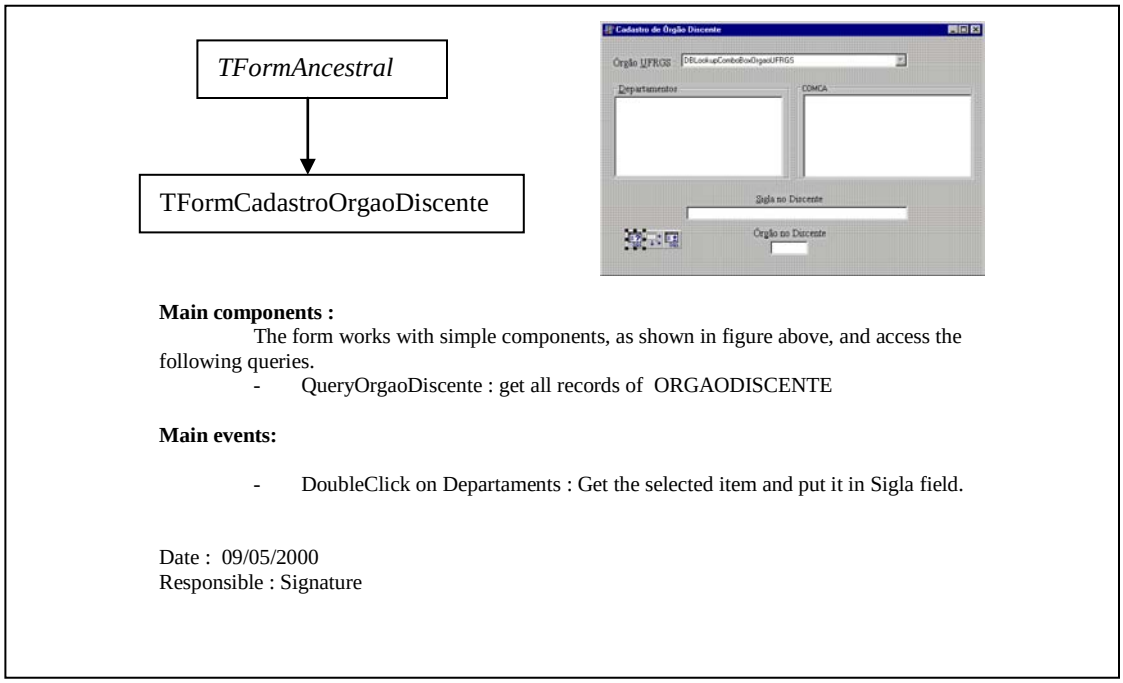


Figure 2. Ex. of Abstract Window Description

*Benefits:* It offers a high-standard vision of the code and turn the communication between programmers and analysts easier. The picture makes the explanation of the window operation easier.

## 4.6.Colors in Obligatory Fields

All mandatory fields need a different color. For our systems, we are using a yellow soft color to show obligatory fields.
*Benefits :* Turn the interface user-friendliness and turn the communication easier. It avoids error messages in case of fields not filled.
Ex.

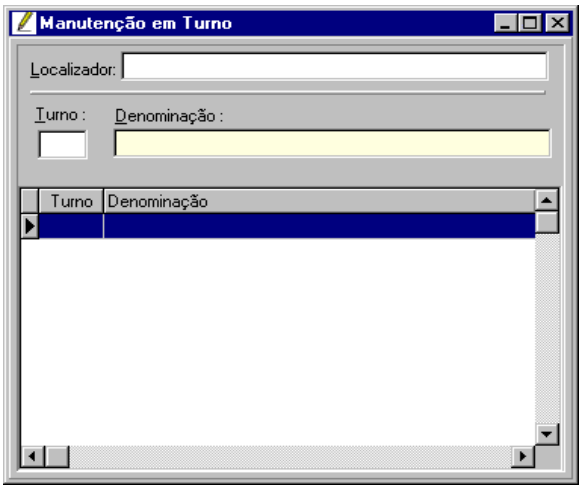

Figure 3. Colors in mandatory fields

## 4.7.Documentation Files Nomenclature

To centralize all documentation, it is very important to use a standard nomenclature in documentation files.
Ex.:

- Description <Window_name>.doc, for window description document
- Guideline <Window_kind>.doc, for guidelines window construction

*Benefits:* easy location of the documentation

## 4.8.Manuals

Two kinds of manuals are needed: programmer and user manual. In the programmer manual there should be the whole documentation about the system. Despite these documents being enough, diagrams are recommended. In the user manual there should be all steps to software installation, configuration and use.
*Benefits:* complete documentation, making the continuity and maintenance of the software easier.

## 5. Walkthroughs

An old concept is the Walkthroughs. They are sporadic meetings with the staffs, to *walk through* a code looking for some mistakes [PAR 86]. Because of the growing complexity and size of the programs, they are not possible nowadays.
But we are proposing sporadic meetings with programmers to exchange ideas and develop software techniques, not excluding the possibility to walk through a system to find mistakes. One programmer is responsible to mark the meetings whenever necessary.
As an advantage of walkthroughs we can mention:

- shared ideas multiply themselves [PAR 86].
- makes the maintenance of patterns easier
- reduce complicated future changes [ART 87]
- reduce corrective maintenance [ART 87]

## 6. Results and Future Work

This paper presented a new methodology for software documentation showing the main benefits using it. As a result of documentation standards we have, initially, the increase of speed in software development.
Walkthroughs are showing good results too. For instance:

- Programmers are more motivated
- The patterns are being more easily followed

Besides, as a result of these meetings we are making a Programming Manual, developed by the programmers and monitored by two analysts. This manual contains all standards of software development, the main programming techniques, tips and error screens, which make this manual different from others and would help new programmers to continue the systems development.
Another result of these walkthroughs is that we are developing components and templates for Delphi Language. The components are very frequently used in our systems and the templates allow create, for example, a main window of a multiple document interface(mdi) system with just one click. Besides, our ancestral classes were placed in a public folder, for all staffs use. Besides that, three new programmers entered on CPD in september and as soon as they were presented to the systems, they received the manual and a presentation about the software development and, two weeks later, they´re completely evolved with the system. That´s a significant result because the time to programmers understand the system decrease in one week, approximately.
We didn´t compare this methodology with another one because we didn´t found that.
Now we have a pattern to be followed and also we can, as a future work, create a change management system, to manage all maintenance occurred in systems. In addition, now we can collect software metrics to analyze results.

## 7. References


[REZ 97] REZENDE, Denis Alcides. Engenharia de Software Empresarial. Editora Brasport, 1997.

[PRE 87] PRESSMAN, Roger S. Software Engineering. A Practitioner's Approach. Mc Graw – Hill Book Company. Second Edition, 1987.

[PAR 86] PARIKH, Girish. Handbook of Software Maintenance. Wiley – Interscience publication, 1986.

[ART 87] ARTHUR, Lowell Jay. Software Evolution. Wiley – Interscience publication, 1987.

[LEA 98] LEÃO, Marcelo. Delphi 4 - Curso Completo. Axcel Books do Brasil, 1998.