WD-pic,
a New Paradigm for Picture Drawing Programs and its Development as a Case Study of the Use of its User's Manual as its Specification

by

Lihua (Lizzy) Ou

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2002

©Lihua Ou, 2002

Abstract

The pic language is a graphics language for specifying line drawings to be typeset. The pic program is a pre-processor of troff that runs in batch mode on Unix environments. In this work, WD-pic, a **W**YSIWYG **D**irect-manipulation **pic**, is developed. WD-pic operates on a new paradigm for WYSIWYG, direct-manipulation picture drawing in which mouse movement is minimized by use of natural defaults being used for information normally provided by the mouse, and in which the internal representation is directly editable in the program. The work is also a case study of using the user's manual for a Computer-Based System (CBS) as its requirement specification. The result of the case study indicates that along several dimensions, user's manual makes an excellent requirement specification for CBSs. The user's manual not only specifies the what not how of the CBS at the users level, but it also serves as a useful requirements elicitation and validation tool, as a repository of use cases, and as a useful source of covering test cases.

Chapter 3

Case study

As mentioned in Section 1.2.1, one goal of this thesis is for it to be a case study in the use of a user's manual of a software product as requirements specification. The goal of the case study is to see how well the user's manual of WD-pic works as the requirement specification and how effectively the manual can be used as a reference in the design, implementation and testing phases. In this case study, Berry worked as the customer of WD-pic. The author of this thesis was the software designer, developer and tester.

3.1 Project plan

The project started in the beginning of October, 2001. It was planned to be finished by the end of July, 2002, for a total duration of 10 months. The project schedule was planned as a classic waterfall, as illustrated in Figure 3.1. The rest of this section quotes the project plan as it was written in September, 2001, including the project schedule, requirements, design, implementation, and testing plans.

Schedule

- preparation 1 month, from October 1 to October 31, 2001
- requirement 2 months, from November 1 to December 31, 2001
- design 1 month, from January 1 to January 31, 2002

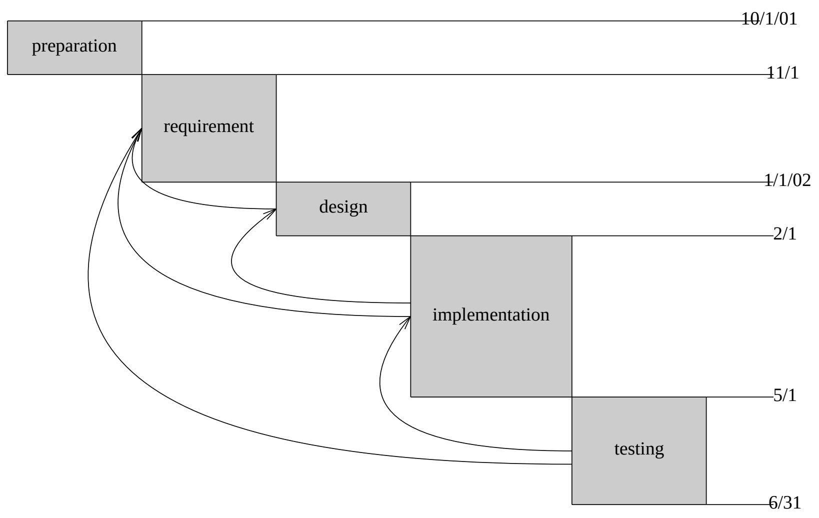

Figure 3.1: WD-pic project plan

- implementation 3 months, from February 1 to April 31, 2002
- testing 2 months, from May 1 to June 31, 2002
- others 1 month

During the whole development process, the manual will be kept up to date.

The preparation phase is to study Shpilberg's prototype of WD-pic and all the existing documents, and to get familiar with pic, its features, its source code, Java Swing, and Jni.

One month flexible time is left to deal with unexpected events.

Requirements

No formal requirements specification will be written. Instead we will start by writing a user's manual, which will be used as the requirements specification. The software engineer will discuss with the customer all the features and document them in the manual. The user's manual will be organized by use cases. It will describe all the fundamental concepts as well and is not to be ambiguous.

Design

All the features in the manual will be designed. High-level design modules should be given. The design should be clear enough to show how the modules work together.

Implementation

All the features in the manual will be implemented. Java Swing will be used to code the GUI. Java Jni will be used to communicate between the Java coded GUI and the C coded pic compiler.

The project will be carried out on a Unix Solaris environment.

Testing

Both black-box testing and white-box testing will be used to test the program. The user's manual will be used as a source of test cases for black-box testing. All the use cases in the manual should be covered.

Unit-testing will be done on every class.

3.2 Results and introspection

The project was carried out, not exactly matching the schedule that was planned. The following is a list of the actual milestones:

- October 2, 2001, project started
- November 1, manual started
- March 28, 2002, manual final (design started)
- April 20, design final (implementation started)
- April 22, manual first update
- May 15, first demo (basic features done)

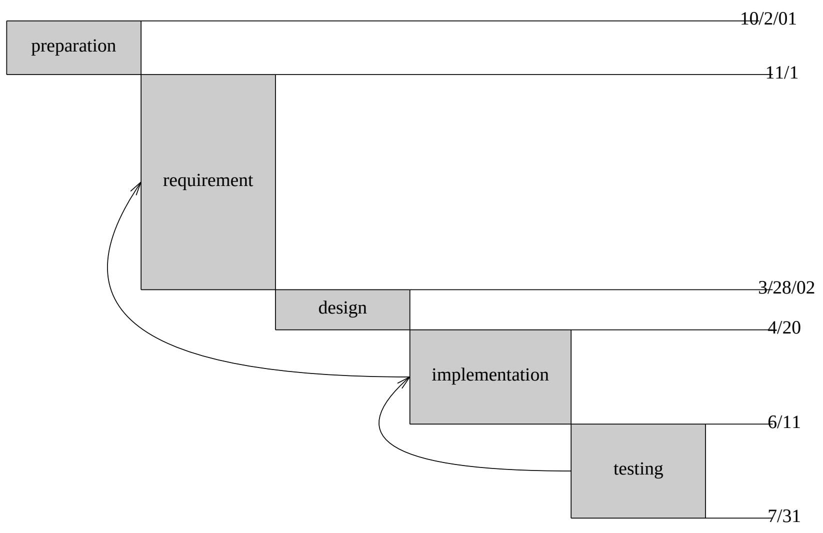

Figure 3.2: WD-pic development process

- May 16, manual second update
- June 11, second demo (all features done)
- July 31, testing end

Figure 3.2 illustrates the whole development process.

Some phases in the software development process are not exactly bounded to the clear time-line. The spare time in the previous phases of implementation was used to do research, such as writing proof of concept (POC) code. The implementation phase was counted from writing the main frame code of WD-pic.

White-box testing was actually combined with the implementation phase. Whenever a class was developed, it was white-box tested. Whenever a feature was implemented, it was tested against the use case in the manual. Thus, the testing phase in Figure 3.2 was for only black-box testing after all the features were implemented, *i.e.*, integration testing and system testing.

3.2.1 Requirements

The requirement phase took longer time than that was planned, totaling 4 months, from writing the outline to finishing up all the features. Writing the manual was actually the process of requirements elicitation. Sometimes, a feature was proposed in a very early version, but it took several revisions to capture what the customer really wanted. In some cases, the customer did not know exactly what he wanted until he had seen the manual description of what he thought he wanted. We had 11 revisions before the implementation started, and 3 more revisions later when the author decided to change the manual from MS Word format to LaTeX format. Some revisions had only minor changes, to correct mostly grammar mistakes.

To avoid ambiguity, the final version of WD-pic user's manual has a glossary defining 34 fundamental concepts including the program name "WD-pic", "user", "grid", and "gravity". The main part of the manual was organized by use cases, from basic use cases to advanced features. A step-by-step sample run was given for each use case.

Later, after implementation started, we had the following several updates.

- We planed to play an alert sound if there was a syntax error when the user tying in the edit window. We decided to use a different color, red, to mark out the text with errors, as this would help the user better than the sound, which is unfocused.

- A status bar was added at the bottom of the main frame to show session command errors. The advantage of using status bar is that the user does not have to click the **OK** button in the warning dialog as in other applications. The coordinates of the cursor on the canvas can always be shown in the status bar as well.

- Selected object is high-lighted in the same selection color as the selected text in the editor window. Then the user can easily see the relationship between the selected object on the canvas and in the edit window.

- It was said in the manual that when the user selected an object on the canvas, the attributes of that object would be shown in the attribute area. But in the implementation, the attribute area is made up of attribute buttons. So it is difficult to show all the values of the attributes in the attribute area. So we changed the attribute buttons to be used for insertion only, *i.e.*, no matter what the existing values are, LMCing an attribute button always resets the value of the attribute in the internal representation.

- There is no grid information in the internal representation. But the customer wants grid to be saved. We need save the grid in a file on the hard disk. During the implementation, we realized that we can further use this file to create a recently opened file history, which is a popular feature in most GUI applications.

- It was said in the manual that preference setting effected inputting from the palette the same as inputting from the keyboard. In the implementation, it is difficult to let the program know what the token is just input from the keyboard. So we changed the preference setting only effects the input from palette. Refer to the user's manual for details.

- We planed to develop and execute WD-pic in a Unix environment. Because the code was written in Java, we built a Windows version without too many changes in the code.

The first three updates actually could have been avoided if we do a better review of the manual, but not the last four. They are related to implementation details.

Compared to writing traditional requirements specification, writing the user's manual at the requirement phase has the following advantages.

- The user's manual is written at the user's level, so it is easy for the user or the customer to see and to tell what he or she wants. It helps requirements elicitation.

- It is also easier for the software engineer to capture what the customer wants. By writing a use-cases-centered user's manual, what the user inputs to the CBS and how the CBS responds to the user are clear to the software engineer.

- By reading the use cases, the customer can verify whether these are what he or she wants.

Berry, the customer's previous experience being a requirement engineer and a customer is relevant to this case study because he learned how to be a demanding customer. But his previous experience with WD-pic was irrelevant, because each time he does this, he goes for what he believes are correct requirements. The fact that he has done WD-pic before changes only set of features not the software engineer's job to find out what he wants and to specify and implement it.

3.2.2 Design

WD-pic user's manual was used as the guideline for design. It is actually a repository of use cases, from basic to advanced. By reading all the use cases, it is not difficult for the designer to figure out the main modules. Then each key use case was visualized to a UML sequence diagram. Last, the user's manual was used to verify whether the designed modules working together to carry out all the use cases. Some of these sequences diagrams are given in Appendix B.

The user's manual helps the design in the following ways.

- The use cases are already there, it is easy for the designer to generate the use case diagrams.
- Because a use case in the manual describes in each step what the user inputs to the CBS or what the CBS responds to the user, it is easy for the designer to construct the sequence (scenario) diagrams.
- The manual helps the designer to verify whether the design covers all the features in the manual by verifying each use case.

3.2.3 Implementation

The user's manual was used as the guideline to implement all the features. Lots of research was done during the process of preparation and requirement, such as studying the pic source code and Shpilberg's prototype code and getting familiar with Java Swing and Jni. POC code was written in the requirement and design phases. For instance, a small piece of Java code was written to show that the C-coded pic compiler works with a simple Java-coded GUI program. Therefore, once the design was fixed, the software engineer could write a main frame and put everything together to make a rough working version, then implement the use cases and abstracted features in the manual one by one. From this rough working version to all features done, it took about two months. Totally, there are 24,091 lines of code (LOCs) in WD-pic. Among all the source code, 13,524 LOCs are GUI code in Java, 10,464 LOCs are pic compiler code in C, and 103 LOCs are external editor code in C. Among the pic compiler code, 9,567 LOCs are reused code from the original pic compiler. 897 LOCs are newly coded. Implementation went much faster than expected.

WD-pic was planned to run only on Unix Solaris systems. However, the result shows that after compiling the pic source code into a dynamic link library on Windows, it works with the GUI in Java on Windows as well. So a Windows version of WD-pic was implemented as well, with slightly different code for font setting and invoking the external editor.

3.2.4 Testing

White-box, unit testing was done to every class during the implementation phase. The user's manual was used as the test plan and the source of black-box test cases. Results show that the user's manual works as a good test plan and source of test cases. The user's manual has all the information that a normal test plan and test cases have, including the system requirement of the program, the execution steps, and the correct results. Once a feature was implemented, the software engineer did black-box testing by following the exact steps in the manual to make sure the program worked as expected.

Most of the testing was done during implementation. Later, more black-box testing was done when the author used WD-pic to draw all the line diagrams in this thesis. The author realized that it was very convenient and fast to use WD-pic to draw the line diagrams if the user had the layout of the whole diagram in mind. Users of the traditional batch pic reported the same phenomenon.

To summarize, the user's manual helps testing by serving as the test plan and providing test cases.

3.3 Author's feelings during the life cycle

When I first heard the idea of using a program's user's manual as its requirement specification, the idea sounded a little bit strange, because in a normal life cycle, the user's manual is the last document to be written. Furthermore, a user's manual and a requirement specification have different readers. They have different focuses on the content.

During the preparation phase, as I was getting to know more about the pic program, I was not sure how much this GUI we were going to built on top of pic could enhance the pic program's functionalities. Even if a user knows the pic language well, the flexibility that the pic program gives to the user is limited because of the limitations of the batch mode program.

The requirement phase was the hardest phase during the whole life cycle. The requirement elicitation was hard. Describing the requirement to capture what the customer really wanted was hard too. I felt that writing a good user's manual which is to be used as a requirements specification is as hard as writing a normal requirement specification. We didn't save time during the requirement phase by writing the user's manual instead of a requirement specification. In fact, I would say that we lost time.

Everything got paid back in the later design, implementation, and testing phases. Having walked through the requirement elicitation process, I knew exactly what I was supposed to implement. Testing became very easy too, as the use-case-oriented manual itself was a source of covering test cases. Implementation went much faster than expected. When I did the first demonstration to the customer, the customer was really impressed.

When I was using WD-pic to draw the diagrams in this thesis, I realized that this software turned out to be much better than I expected. The GUI gives users much more flexibility to draw diagrams than the original pic program. Without too much knowledge of the pic language, I was able to draw all kinds of line drawings easily.

Compared to projects that I did before, the requirement phase in this case study was no easier than that in a normal life cycle. I expected that writing a user's manual would have been easier than writing a formal requirement specification. However, design, implementation, and testing went much better than expected. The project finished on time and with the customer's satisfaction. While in my past experience, of about 5 years, usually the early requirements and design phases went much more smoothly than in this project. However, in the past projects, always requirements and design problems were discovered during implementation and testing. In this project, there were much fewer problems discovered during implementation and testing, allowing them to go very quickly.

Chapter 4

Conclusions

In this work, two contributions are that,

1. WD-pic was implemented and
2. a case study of using a program's user's manual as the program's requirement specification was carried out.

We have the following conclusions.

First, WD-pic follows a new paradigm for WYSIWYG direct-manipulation picture drawing programs. It has the following advantages.

- It inherits all the advantages that the batch mode pic program has and it fixes the disadvantages of the batch and most WYSIWYG picture drawing programs.
- It provides convenient direct manipulations and directly editing the internal representation of picture to users.
- Input by mouse or by keyboard are fully interchangeable. The user does not have to inform the application where its next input is coming from.
- It minimizes the mouse movement.

However, because WD-pic is built on top of the pic program, its features are limited to the features that pic can provide. Besides this, the current implementation of WD-pic also has some other shortcomings that are independent of pic.

- Changing the sizes and locations of objects cannot be done by direct mouse manipulations.

- Copy, paste, cut, undo, and redo do not work with the canvas.

- Changing attributes of constructs cannot be done by direct manipulations.

- Pictures cannot be output in a standard graph format.

The first two are not real limitations. They can be overcome by using the built-in text editor. They are not difficult to fix based on the current implementation. The later two are the challenges for the designer of the next version.

Second, the result of our case study shows it is useful to write the user's manual at the requirement phase. A non-ambiguous, use-case-centered user's manual helps the whole process of the software development.

- The user's manual makes an excellent requirement specification for CBSs. It specifies the what-not-how of the CBS at the users level.

- It helps requirements elicitation by helping both the customer and the software engineer to see what is wanted. But it cannot solve the problem that sometimes, the customer does not know what he or she want.

- The user's manual is a good validation tool; it helps the customer to verify the requirements specification, and helps software engineer to verify the design and implementation.

- The user's manual as a repository of use cases, and a useful source of a test plan and covering test cases.

- There is no need to write the user's manual again after the development finished! The one used by the software engineer a lot is also easy to be read by users.

There is no completely satisfactory way to validate any SE method, but at best, our result shows that using a program's user's manual as its requirement specification is a promising method. It is worth additional case studies.