Software Workshop Team Java (06-08165) 2006/07, Dr. S. Vickers

Project Report: An Applet to Demonstrate ...

Team X:
Fred Flintstone
Wilma Flintstone
Barney Rubble
Betty Rubble
Dino

February 15, 2007

Work Breakdown

Coding	**Dino (Team Leader)**	**Fred**	**Wilma**	**Barney**	**Betty**
Datastructures	Contribution: 100%	0%	0%	0%	0%
...	...	...	...	...	...

Report	**Dino (Team Leader)**	**Fred**	**Wilma**	**Barney**	**Betty**
Introduction	Chapter 1: 100%	0%	0%	0%	0%
...	...	...	...	...	...

Contents

Abstract

You should write a one-page abstract written as an "Executive Summary". It should be written for someone who is familiar with the Team Java module, so that there is no need for background or generalities. Rather, you should explain what is special about your project, and what you claim to have achieved. (One page maximum.)

Chapter 1

Introduction

Give a brief overview and guide the reader to the important points in the remaining sections.

This template for your report also contains some examples of how to use some LaTeX elements and commands. In particular, there are examples for tables, how to include figures, and various environments for bullet points or enumerations.

For further information (and here is how to do a bullet list):

- look on the Team Java web page,
  `http://www.cs.bham.ac.uk/internal/courses/team-java/current`
  (Click on "Guidance".)
- google "latex"
- look at the LaTeXbook [1]

This is how to do numbered lists:

1. First point
2. Second point
3. ...

This is how to do sections:

1.1 Some topic

1.2 Another topic

If you set a label with the `label` command, you can then use the `ref` command to refer to that section – e.g. section 1.1. This means you don't need to know about section numbers. Note that ~ means a space that cannot be broken across lines.

Chapter 2

Requirements

You have been given only a very general set of requirements; hence you have to formulate more specific and detailed user requirements for the applet you chose to do and explain why you chose these features. Explain your choices carefully (not just lists of bullet points). Note that not everything about requirements in general is relevant for this project (for instance, you need not write much about the hardware requirements, as they are trivial for this project).

Chapter 3

Design

This section is crucial. Describe the overall structure of your program at a suitably high level of abstraction. For instance, UML diagrams or informal box-and-arrow diagrams can be used to describe program structure. Be sure to describe the MVC structure used. Note that code listings or screenshots are not appropriate here. An important point is how you have divided the project into modules that different team members can work on, and how these are then integrated. For example, you could use interfaces to describe a clean boundary between modules, so that some team members use the functionality provided by the interface, while another team member implements it. Bear in mind Software Engineering principles of good design like coherence and coupling.

Chapter 4

Validation and Testing

Your applet is expected to be in good working order and do something useful. This chapter is important, because it describes how you assure yourself that that is the case.

Testing is so you can be confident that the software is robust and bug-free. What was your strategy for that? How did you plan unit testing (for components) and integration testing (for the whole applet)? How did the prototype fit in?

Validation is to check that in the end the applet is useful and pleasant to use. What was your strategy for that? Have you tried it out with teachers or students? What kind of rolling validation did you use for the evolutionary part (developing the GUI)?

Establish what your project can handle successfully, and what its limitations are. Use meaningful examples, not lists of trivial cases.

Chapter 5

Project Management

The week-to-week management of your project is documented in your weekly progress reports, so you do not need to repeat that information here. A number of important points that you need to address are:

- What are the main components of the project (or *workpackages*) that have to be completed for it to succeed?
- What software process model (e.g. waterfall or evolutionary) did you use for *individual workpackages*?
- How did you allocate team members to particular tasks?
- How did you coordinate what team members were working on?
- How did you communicate the relevant technical information in the team?

Note that a clean design makes all of these easier, so you could refer to your design section, i.e. chapter 3, where appropriate. You can also discuss difficulties in the team interactions if there were any, and how you dealt with them.

You can, for example, include a chart detailing the time management of your project as shown in figure 5.1. Explain the single workpackages in the text:

Workpackage 1: Requirements specification (Week 1)

Workpackage 2: Something else... (Week 2—3)

Workpackage 3: ...

This is only an example chart, of course; your own timeplan will probably look differently. Note that different workpackages can be worked on in parallel!

Figures are included in LaTeX as `.eps` files. In Linux you can use `convert` to convert any type of graphics file to `.eps` format.

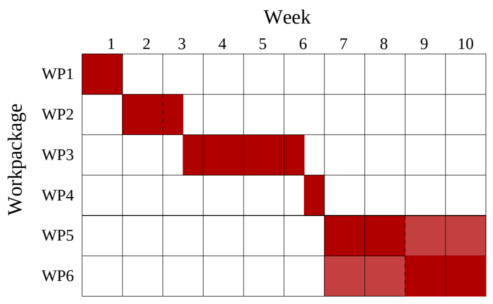

Figure 5.1: Example timeplan of a project.

Chapter 6

Conclusions

Evaluate what you have achieved in your project in objective terms (not just things like “we are all happy with the result”). What are the strenths and limitations of your project? If you had more time, what could you add? If you could do it all over again, what would you do differently? Are there general things about software or team management that you have learnt in this project that you could apply if you were going to work on a (perhaps completely different) team project in the future?

Bibliography

[1] Leslie Lamport. *LaTeX: A Document Preparation System: User's Guide and Reference Manual.* Addison-Wesley, 1994.

[2] Kevin Lano, Jose Luiz Fiadeiro, and Luis Andrade. *Software Design using Java 2.* Palgrave Macmillan, 2002.

[3] School of Computer Science, University of Birmingham. Guidance notes on plagiarism. `http://www.cs.bham.ac.uk/internal/students/plagiarism.htm`.

Acknowledge any code you have used (if any), and also any that was generated automatically, e.g. by wizards.

Give correct and complete bibliographic information for any sources cited. See the local referencing guide. For instance cite which books on software engineering you have used, for instance [2]. Use the `report.bib` file to manage your citations.

When you *quote* material from other sources, you must be absolutely clear *at the point where you quote it* exactly which of your material is quoted and what the source is. As explained in [3],

"Direct quotation is not particularly common in scientific writing, as it is generally not the words that matter, but the meaning. Normally it is preferable to rewrite someone else's ideas in your own words, often changing the terminology and other superficial details to suit the new context.

However, in circumstances where it is appropriate to make direct use of the words of another person, those words should normally be included within quotation marks and a reference to the source of the words given in the usual way."