Table of Contents

Tips, Tricks and Typesetting Style *

Firstname Firstsurname [1], Secondname Secondsurname [1], and Thirdname Thirdsurname [2]

[1]Technical University of Košice, Department of Computers and Informatics
Letná 9, 042 00 Košice, Slovakia

[2]Slovak University of Technology, Faculty of Informatics and Information Technologies
Institute of Informatics and Software Engineering
Ilkovičova 3, 842 16 Bratislava 4, Slovakia

Abstract. Abstract should express the orientation and main contribution of your work (not your thesis) in 6 lines. The aim of this paper is to present the essential rules when typing source LaTeX files and to prevent the most frequent mistakes. Comparing source `tipsandtricks.tex` and target `tipsandtricks.pdf` is the best approach to acquire useful information.

Key words: Software Engineering, refactoring, ... data driven transformation. (Maximum of six keywords, starting with the most general, and finishing with the most specific).

1 Coding and Slovak Accents

1.1 Preparing English Paper

When preparing an English paper for Proceedings, you must use CP1250 coding, since then separate papers from different authors can be simply composed.

You must use international accents in special form, for these four extraordinary cases: `\softt` to get ť, `\softl` to get ľ, `\softd` to get ď, `\softL` to get Ľ. Do not forget to add one space character after each of these commands.

You must also use standard international accents, such as `\v{c}` instead of `č` to produce correctly typed č, `\'\i` instead `í` to produce correctly typed í, `\"a` instead of `ä` to produce correctly typed ä, etc.

Never use slovak package in English papers, otherwise you will get some sections in slovak language.

1.2 Preparing Slovak Documents

When preparing separate Slovak documents, such as your Thesis, CP1250 coding combined with slovak package is recommended, introducing

* This form of acknowledgement is sometimes required, but here should be excluded

```
\usepackage[cp1250]{inputenc}
\usepackage{slovak}
```

in declaration part of the document. Other than CP1250 coding is certainly possible for a single document, but babel package is not recommended.

In Slovak documents, the use of international accents in .tex source is not necessary and Slovak QWERTY keyboard can be used. However, it is good idea to check correct form of all Slovak letters with accents before preparing your document.

2 General Typesetting Rules

Paragraphs should be separated by empty line, not by hard line terminator – double backslash `\\`.

Just when finalizing your document, if some text exceeds the right margin, use `\newline` (not `\\`).

Instead of single dash `-` you must type `--` in LaTeX .tex file, to get correct dash form – in the target .pdf file.

Don't use `\noindent`, `\vspace{}`, `\hspace{}` aiming to 'improve' the design of your PDF document. Accept LaTeX typesetting defined by defined style.

3 How to translate source LaTeX file and view the target PDF file

Translate your source LaTeX file (.tex) using `LaTeX=>PDF` profile pressing icon **Built current file (Ctrl F7)**. You will get PDF file.

View produced PDF file by pressing icon **View Output file (F5)**.

Very important: Do not forget to translate the final version at least three times and check unresolved references – by searching question mark ? in produced PDF file.

4 How to citate and to refer

In citations, use your surname as prefix, i.e.
`\cite{surname:Gurp,surname:Greenfield}`, where surname is your surname, not just `\cite{Gurp,Greenfield}`. Corresponding to this citation, bibitems `\bibitem{surname:Gurp}` `\bibitem{surname:Greenfield}` in References must exist. Each bibitem should be citated in text, otherwise it is irrelevant. By other words, you can introduce in the list of references just titles that you citate in the text of your document.

As an example of citation, automatic software construction by machines is software engineering vision [6,5]. Generative [2], aspect oriented [11], or context oriented programming [10] are ...

The same holds for referencing: Use `\ref{surname:Program}` when referencing something labeled by `\label{surname:Program}`

5 Your Own New Commands and Inline Maths

Prevent your own definition of new commands, if possible. If you still define your own definition, such as in this document

```
\newcommand{\lp}{\mbox{ $[$\hspace{-1.4pt}$[$ }}
\newcommand{\rp}{\mbox{ $]$\hspace{-1.4pt}$]$ }}
.....................
```

you can use defined commands `\lp`, `\rp`, ... see below.

The execution in G should be semantically equivalent to P, i.e. $P = G\ S$, where $[\![P]\!] = [\![S]\!] + [\![G]\!]$.

Inline math symbols and inline math formulas are enclosed in `$` signs, i.e. P is typed as `$P$` and $P = G\ S$ is typed as `$P\ =\ G\ S$`. Note: mathematical font differs from textual font and space in math mode is marked by backslash space characters.

6 Using Numbered Formulas and Equations

In general, all formulas and equations should be numbered.

Let us define a simple grammar, as follows.

$$\begin{array}{ll} \mathit{Language} & \rightarrow \text{print } \mathit{Exp} \\ \mathit{Exp} & \rightarrow \mathit{Mul}\ \underline{\mathrm{C}_1}\ \{\ \underline{\mathrm{S}_1}(\ +\ |\ -\ )\ \mathit{Mul}\ \} \\ & \cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \mathit{Mul} & \rightarrow \mathit{Term}\ \underline{\mathrm{O}_1}\ [\ \underline{\mathrm{S}_2}(\ *\ |\ /\ )\ \mathit{Mul}\ ] \end{array}$$

Let us define grammar (1). This numbered form is preferred.

$$\begin{array}{ll} \mathit{Language} & \rightarrow \text{print } \mathit{Exp} \\ \mathit{Exp} & \rightarrow \mathit{Mul}\ \underline{\mathrm{C}_1}\ \{\ \underline{\mathrm{S}_1}(\ +\ |\ -\ )\ \mathit{Mul}\ \} \\ \mathit{Mul} & \rightarrow \mathit{Term}\ \underline{\mathrm{O}_1}\ [\ \underline{\mathrm{S}_2}(\ *\ |\ /\ )\ \mathit{Mul}\ ] \end{array} \tag{1}$$

Let us define the rule, as follows.

$$\mathit{Language} \rightarrow \text{print } \mathit{Exp}$$

Let us define a rule (2). This is numbered, i.e. preferred form.

$$\mathit{Language} \rightarrow \text{print } \mathit{Exp} \tag{2}$$

7 Using Fonts

7.1 Using Typewriter Font

It is better to type `WHILE` (by `$\tt WHILE$`) than WHILE (by `\verb/WHILE/`), when using typewriter font inline.

Prevent using verb and also verbatim environment, especially when typing programs.

Introducing program fragment as follows is inappropriate and not acceptable.

```
private int highShortOf(float value) {
  ByteBuffer buffFloatInt = ByteBuffer.allocate(12).putFloat(value);
  return buffFloatInt.getInt(0) >> 16 & 0177777;
}
```

Program fragments should be included in figures, since just then *they can be referenced*, see subsection 8.1.

7.2 Changing Textual Fonts

As a general typesetting rule, fonts should be changed exceptionally.

New (and preferred) style for changing textual fonts is **alfa**, *alfa*, `alfa`, alpha, obtained by

```
\textbf{alfa}, \textit{alfa}, \texttt{alfa}, \textsf{alpha}
```

Old style style for changing textual fonts **alfa**, *alfa*, `alfa`, alpha, is also supported by

```
{\bf alfa}, {\it alfa}, {\tt alfa}, {\sf alpha}
```

7.3 Changing Math Fonts

Changing math font is sometimes useful, for example `${\bf print}\ E$` (in oldstyle) or `$\mathbf{print}\ E$` (in new style) both produce **print** E.

Decision for appropriate environment is important – tabular environment for the most frequently used text fonts or array environment for the most frequently used math fonts. Good decision minimizes additional font changes for environment items.

7.4 Emphasizing

You can use *slanted form of emphasizing your text* (`{\em ... }` is preferred) or less recommended **boldfaced form of empasizing your text** (`{\bf ... }` but never mix them in one document.

Never use colors to emphasize text in your document.

8 Figure Environment

8.1 Program fragments in figures

You should introduce program fragments in figures, such as in Fig. 1.

Or, you can decide for other style, such as in Fig. 2

Use just one style for programs according to your decision, don't skip between different styles in your document.

```
private int highShortOf(float value) {
    ByteBuffer buffFloatInt = ByteBuffer.allocate(12).putFloat(value);
    return buffFloatInt.getInt(0) >> 16 & 0177777;
}
```

Fig. 1. Java Program Fragment – first style

```
private int highShortOf(float value) {
    ByteBuffer buffFloatInt = ByteBuffer.allocate(12).putFloat(value);
    return buffFloatInt.getInt(0) >> 16 & 0177777;
}
```

Fig. 2. Java program fragment – second style

8.2 Pictures in figure environment

When including pictures (.jpg, .png, .pdf), such as screenshots or your own drawn pictures in figure environment, it is recommended to work with known size in mm, since additional resizing may result to invisible textual description, and to prove appropriate density (it should be at least 180DPI, but sometimes 400DPI is required).

Therefore, it is highly recommended to try printing simple picture sample before drawing all pictures.

An example of a not very good picture is in Fig. 3. Although it has good density (400DPI), and it seems quite well drawn, it has some design errors, enumerated below.

The main design errors of picture in Fig. 3 are as follows.

1. There is two much free space and too small boxes and graphic shapes in this picture. As a result, text and graphic shapes are overlapping – this is disturbing.
2. Textual descriptions are unsharp and text size is too small. This decreases good impression and it is not informative, hence it is not acceptable. The size of text in picture descriptions must be approximately of the same size as in text of document. It must not be larger, it may be smaller, but not tiny.

Very important Since source file `tipsandtricks.tex` *and picture* `tippicture.jpg` *are in the directory* `tipsandtricks` *and source is processed from parent directory, you MUST use path* `tipsandtricks` *for picture, as follows*

```
\includegraphics*[scale=1.0]{tipsandtricks/tippicture.jpg}
```

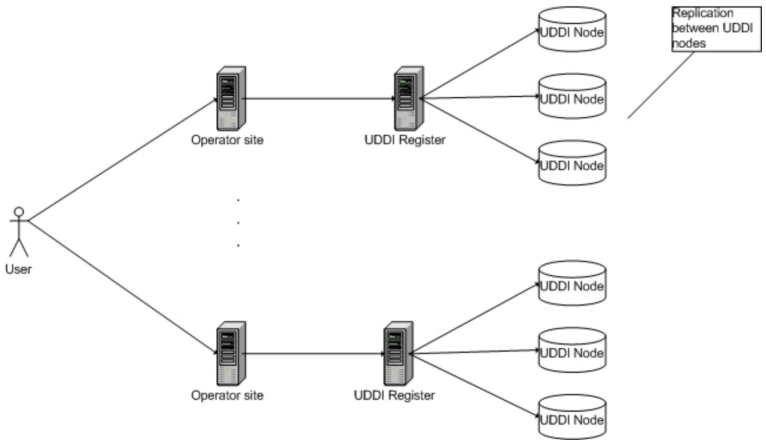

Fig. 3. Stream driven quiet computation

Including picture without path, as follows

```
\includegraphics*[scale=1.0]{tippicture.jpg}
```

is not correct.

8.3 Positioning figures

You must introduce figure environment (the same holds for table environment), immediately after referencing it. Essentially a figure position is given by `[h]` (here), `[t]` (top of this page), `[b]` (bottom of this page), and `[p]` (on the separate next page) qualifiers. Recommended combinations are `[h]` (here), `[htb]` (anywhere in this page), and `[p]` (next page with single figure).

Figures should be positioned in a section, in which they are referenced, sometimes (since of automatic LaTeX figures positioning) they are shifted somewhere to the end of your document. In this case, command

```
\clearpage
```

introduced in separate line is **very useful**.

Changing the position of figures manually until document is finished is not very good strategy.

9 Using enumerate, itemize and description environments

Using itemize and enumerate environments is preferred, for example

– If C_k consumes 1, ...

– If O_k consumes 1, ...

1. If C_k consumes 1, ...
2. If O_k consumes 1, ...

Description environment should be used occasionally

Rule 1 If C_k consumes 1, ...
Rule 2 If O_k consumes 1, ...

Prevent too complicated hierarchical structure of environments, such as

Rule 1 If ...
1. If it consumes 1, ...
   – Then follows ...
   – Then follows ...
2. If it consumes 0, ...

Rule 2 If ...

10 Analogy of Environments and Caption Positioning

Tabular environment is a textual analogy to array environment (usually used in math mode in equation environment, but also enclosed by dollar signs as a tabular item).

Table environment is an analogy to figure environment – both have captions.

In figure environment, caption should be under the figure in both Slovak and English documents.

In table environment, caption should be placed under the table in Slovak documents but over the table in English documents.

11 Acknowledgement

This is important section of your paper and should be either left unchanged or changed on demand of your supervisor.

12 Conclusion

Conclusion should summarize in a few lines your results that have been reached in your work and scientific or application contributions. In this paper, please notice the next section – References: unified form of author's names, and complete information about sources including pages are necessary.

References

1. Črepinšek, M., Mernik, M. Inferring Context-Free Grammars for Domain-Specific Languages, Conf. on Language Descriptions, Tools and Applications, LDTA 2005, April 3, 2005, Edinburgh, Scotland, UK, pp. 64–81.
2. Czarnecki, K., Eisenecker, U. E. Generative Programming: Methods, Tools, and Applications. Addison Wesley (2005), 832 pp.
3. Czarnecki, K., Helsen, S. Feature-based survey of model transformation approaches, IBM Systems Journal, Vol.45, No.3, 2006, pp. 621–645.
4. Sunyé, G., Ho, W., Le Guennec, A. and Jézéquel, J. M. Using UML Action Semantics for executable modeling and beyond, Proceedings of the 13th Conference on Advanced Information Systems Engineering, 2001, Springer, pp. 433–447.
5. Greenfield, J., Short, K., Cook, S., Kent, S. Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. Wiley, 2004, 500 pp.
6. van Gurp, J., Bosch, J., Svahnberg, M. On the notion of variability in software product lines. In Proceedings 2nd Working IEEE / IFIP Conference on Software Architecture (WICSA), 2001, pp. 45–54.
7. Zhao, J., Liu, S., Wang, X., Chen, L., Wei, C. Research and design of an executable modeling language based on MOF, IEEE 9th International Conference on Computer-Aided Industrial Design & Conceptual Design, Kumming, China, 22-25 Nov. 2008, pp. 399–404.
8. Javed, F., Mernik, M., Gray, M., Zhang, J., Bryant, B. R. Using a Program Transformation Engine to Infer Types in a Metamodel Recovery System, Acta Electrotechnica et Informatica, Vol.8, No.1, 2008, pp. 3–30.
9. Lämmel, R. Grammar Adaptation, In Proc. Formal Methods Europe (FME'01), volume 2021 of LNCS, Springer-Verlag, 2001, pp. 550–570.
10. von Löwis, M., Denker, M., Nierstrasz, O. Context-oriented programming: beyond layers, Proceedings of the 2007 international conference on Dynamic languages, ACM International Conference Proceeding Series; Vol. 286, Lugano, Switzerland, pp. 143–156.
11. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, V. G. An Overview of AspectJ. ECOOP'01, 2001, LNCS, vol. 2072, pp. 327–355.
12. Kleppe, A. A Language Description is More than a Metamodel. In: the 4th International Workshop on (Software) Language Engineering. 2007.
13. Klint, P., Lämmel, R., Verhoef, C. Toward an Engineering Discipline for Grammarware, ACM Transactions on Software Engineering and Methodology, Vol.14, No. 3, July 2005, pp. 331–380.
14. da Cruz, D., Berón, M., Henriques, P. R, Pereira, M. J. V. Strategies for Program Inspection and Vizualizations, Proc. CSE'2008 International Scientific Conference on Computer Science and Engineering, Sep.24–26, 2008, High Tatras Stará Lesná, Slovakia, pp. 107–117.
15. Wu, X., Roychoudhury, S., Bryant, B. R., Gray, J. G., Mernik, M. A Two-Dimensional Separation of Concerns for Compiler Construction. Proceedings of the 2005 ACM symposium on Applied computing, 2005, pp. 1365–1369.