Ingredients

Reading: This note is largely inspired by my personal experience and Chapter 9 of Zobel's *Writing for Computer Science* [6], which is a generally good resource about writing for research students (including MSc thesis students). This the only book I know that is written with computing writers in mind, as opposed to aiming at social science students, or humanities students. Recommended! Our library should have a few copies.

If you need to learn more about evaluation and experimentation then read sections 6.7–6.8 in [5]. Otherwise this note should be sufficient for most projects in this course (and for most of your theses).

If you want additional inspiration on reading then google for tutorials (for instance "how to read a research paper in computer science"). In general, however, I think it is more important to practice reading by reading actual research papers, than reading papers on how to read research papers.

1 Introduction

Writing should start before the project's half way mark. Writing will structure your work. You will discover missing data early. You will uncover problems earlier. It will become more clear what work is necessary (because it influences the paper), and what is of secondary importance. Now it is a good time to start devoting resources to writing.

Recall that so far we have already discussed how to define a research problem (writing a project proposal) and that it is very important to have an idea what your claims will be and how you would provide the evidence for your claims.

This is the structure of the report that I suggested (see an earlier lecture note for an explanation):

1. Introduction
2. Background
3. *Real stuff comes here*
4. *Describing your technical solution*
5. *Or performing some analysis*
6. Evaluation
7. Threats to Validity / Discussion
8. Related Work
9. Conclusion

2 Scoping

Write down on a whitebaord or a piece of paper your results, small and big included. Identify the main result and organize the story around it.

- Which results are most surprising and interesting?
- Which outcomes are most likely to be used by someone?
- Which are the outcomes that do not justify publication? Are not worth of the space?
- Which data in experiments supports conclusions? Most experiments produce much more data than you can publish.

(list adapted from [6])

Then you repeat the same exercise as discussed a few weeks earlier: align the problem, the solution, and evidence speaking for this solution. It is inevitable to do small adjustments at the writing stage, as the course of the project very often invalidates your initial plan. Also very often we observe that the original scope was too ambitious, and one can write a better paper for a narrower scope.

3 Telling A Story

Zobel distinguishes four kinds of story:

- A *chain* is the most common structure in software engineering (and also in many theory papers). A chain is a simple story, starting with the problem and its significance, through development of the solution, to providing evidence for success (evaluation).
- In a theoretical paper it is often better to organize *by specificity*. You describe the story first on a simple framework, and then again on a complex one.
- Another possibility is *by example*, where you first present the problem solved by example, and also explain how the solution works by example. The presentation of the detailed algorithm is delayed until the later sections of the paper.

Most importantly it is inappropriate to structure the paper as a log of your work or of your thought process. It is not suitable to describe all the failed attempts, and conceptual improvements you made on the way. It is also not suitable to just report the experiments you have done in chronological order. All these approaches almost inevitably lead to unreadable papers.

Finally, always assume that most readers will not read your paper in detail (later in this note, we advise you to *avoid* reading most papers in detail!). So it is very important that the main points are visible early. This means that the crucial points need to be made in the title, in the abstract, and in the introduction. This also means that in each section, and indeed in each paragraph, you should strive to place the most important message of the paragraph in the first sentence. You should also reserve special typographical means such as emphasis, narrower text, enumerated and itemized lists, bold and *figures* (!) to the most important aspects of the story. Indeed, many authors believe that the main story should be possible to be grasped from just skimming through the figures in the paper.

In any case, if you succeed to draw readers' attention to the strongest points of the paper, there is higher chance that they would read it in detail.

For most unexperienced writers, it is a very good technique to write quickly. You jot down the outline for a section as bunch of points on a piece of paper. Then you quickly write the section following these points, as your thoughts dictate it. It is crucial not to perfect the write up. For many it is best to even avoid correcting grammar and spelling mistakes at this stage.

Simply press onwards without stopping, until you are done with a section, or its significant part. But do follow the outline, otherwise you will end up with a mess!

Do not stop to create figures or diagrams, just leave space for them, and refer to them as if they already existed. In any case, it is often very practical to delegate creation of figures to one of your co-authors.

If you are susceptible to writer's block, then definitely follow the above advice.

After the first draft is created this way, it is normally quite fast to improve in 2-3 sequential editing iterations. Often it helps a lot to read the text aloud when editing. Many people hear much more errors, and imperfections than they can see.

4 Abstract

Abstract is the executive summary of the entire paper. As a summary it should encompass not only the main result, but also the problem, and the evaluation. Shaw [4] reports that good ICSE papers discussed evaluation already in their abstract. This is important in software engineering, because of the emphasis put on the evaluation of results.

A good abstract is readable for non-experts, as it increases the chances that someone will build on your work (often application opportunities appear in other areas).

> I try to have four sentences in my abstract. The first states the problem. The second states why the problem is a problem. The third is my startling sentence. The fourth states the implication of my startling sentence.
>
> Kent Beck in [2]

Simon Peyton Jones summarizes this idea in the following points:[1]

- State the problem
- Say why it's an interesting problem
- Say what your solution achieves
- Say what follows from your solution

Here is an example that I invented on the fly:

> Feature modeling is an important element of development of Software Product Lines. Much research exists on feature modeling languages, and feature modeling tools. It is surprising though, that no experience reports exist of using feature modeling in industrial practice, and no characteristics of industrial models is openly available.

[1] research.microsoft.com/en-us/um/people/simonpj/papers/giving-a-talk/writing-a-paper-slides.pdf

> In this work we identify, present, and analyze two real life feature models of operating system kernels, eCos and Linux. We find that many assumptions about feature models made by tool builders in academia do not hold in reality, and thus many tools are potentially not useful; real models are substantially larger, and with higher density of constraints than previously anticipated. 109 words

It is not important to stick to the four sentences. Often it is easier to write two or three sentences for each of the four parts of the abstract. It is however recommended to stick to the four part structure, at least for your first draft.

Normally we avoid citing other papers in abstracts, since the abstracts get published in various websites and publication indexes, without the list of references attached. We only do this if the paper is improving directly on, or discussing in detail, some other paper.

Abstract is often limited to 150–200 words. In this project the abstract must be at most 150 words.

5 Introduction

Introduction is often very similar to the abstract. Sometimes people write the introduction by expanding a draft abstract into a section. Sometimes, we write the introduction, and then derive the main sentences from it to form the abstract.

In the introduction include the following parts:

- General statement introducing the area; You can most likely start with the first paragraph from your project description and evolve it.
- Explanation of the specific problem and why do we care about the problem.
- Explanation of your solution, and how it improves on the work by others. Relation to related work can be very brief, given that you have a separate extensive section devoted to this.
- A hint on how the solution was evaluated and what was the outcome of this evaluation.
- A summary (a “map”) of how the paper is organized.

An introduction must be very reader friendly, assuming little prerequisites. Thus using technical jargon, complex terminology, mathematical formulae, and alike are discouraged in the introductions.

It is best to delay writing the abstract, conclusion and the introduction to the late phases of writing. Begin with writing the middle sections of the paper. Unless, you are really lost with what the main message of the paper should be — then often starting with an abstract (or an introduction) can help in scoping.

6 Evaluation

Section 6 (Evaluation) often takes form of an experiment that empirically demonstrates the quality (or preferably the superiority) of your result. Sometimes not just the evaluation, but the main body of the paper (sections 3–5) is empirical — if your objective is to study some existing phenomenon.

In such case you are obliged to discuss any possible weaknesses and limitations of the experiment and the limitations of the conclusions that one can draw out of it. These limitations take the name of *Threats to Validity*, and are always placed in the back part of a paper, directly after the experimental section; Section 7 in our example outline above.

In this note we discuss two main classes of such limitations and give example quotes from existing papers.

7 Threats to Validity

The two main sources of threats to validity in an experiment are the construction of the experiment and the generalizability of the results. In other words whether the results are correct for the population (sample) used in the experiment, and whether one can generalize them to a broader set of subjects. The former are called *internal* threats, the latter are called *external* threats.

The literature also includes other kinds (primarily *conclusion validity* and *construct validity*), but for our practice, and for many professional researchers, it will suffice to focus on internal and external threats. However, if you find it difficult to categorize the threats in your project into these two groups, you may want to study the more detailed classification; see for example [5].

First a bit of basic terminology on experimentation:

Subjects is the objects, or persons, being studied, whose properties or behavior is analyzed through the experiment. Subject and objects are sometimes used interchangeably, and some authors would prefer to reserve subject for human who are active in the experiment, and object for artifacts being manipulated (for example models, or program code).

Factor is the property, or technically a variable, whose correlation we study. For example in an experiment measuring speed of programming in C vs speed of programing in C#, the choice of programming language is a factor (and the speed of programming is another variable).

Treatment is one possible value of a factor, so in the above example C and C# will be two treatments.

Outcome is the result of the experiment, so some numbers, other data, or qualitative information, indicating some correlation between treatments and observed variables — in the example the correlation (or lack of thereof) of speed of programming and the choice of programming language.

7.1 Internal Validity

Internal Validity is concerned with confirming that the correlation between the treatment and the outcome is indeed casual, and not accidental, or caused by some third variable that has not been observed. For example we may discover that all programmers in C were faster than programmers in Java, but forget that all the programmers in Java took the experiment very late at night, when they were tired.

Thus internal threats to validity will often be related to the internal construction and execution of the experiment.

Main internal threats appearing in experiments with human subjects:

- History — when two treatments are applied to subject in order at different times. Subjects can be tired or change perspective due to this ordering.
- Learning effects — subjects can learn the task at hand over time. For example if the same person is first asked to implement a program in C, and then the same program in Java, she will likely be able to exploit the experience of solving the task in C to be more effective when working in Java.
- Testing — subjects should not know the results of the test on the fly (the 'election poll' effect)
- Mortality — subjects dropping out during experiment. You need to characterize whether this has not changed the representativeness significantly.

Main internal threats appearing in automatic experiments

- Intrusive instrumentation — the measurement or observing a program changes the variable being measured (for instance speed).
- Statistical insignificance (also in human experiments) — the sample is too small. Check experimentation handbooks if you want to design a statistical experiment.
- Direction of correlation — whether A causes B, or B causes A, or perhaps there are tertiary reasons that cause both A and B.

7.2 Examples of Threats to Internal Validity

The following comes from a paper that gathers a lot of statistics about the variability models of the Linux Kernel and of the eCos operating system:

> An internal threat is that our statistics are incorrect. To reduce this risk, we instrumented the native tools to gather the statistics rather than building our own parsers. We thoroughly tested our infrastructure using synthetic test cases and cross-checked overlapping statistics. We tested our formal semantics specification against the native configurators and cross-reviewed the specifications. We used the Boolean abstraction of the semantics to translate both models into

> Boolean formulas and run a SAT solver on them to find dead features. We found 114 dead features in Linux and 28 in eCos. We manually confirmed that all of them are indeed dead, either because they depended on features from another architecture or they were intentionally deactivated. [1]

Note that the paragraph has the following structure: first it gives the threat (here incorrectness), then explains what have we done to minimize this threat. You should always follow this pattern.

In another paper, we have studied evolution of the Linux Kernel model over time. Here is the internal threat paragraph from that paper:

> Git allows rewriting histories in order to amend existing commits, for example to add forgotten files and improve comments. Since we study the final version of the history, we might miss some aspects of the evolution that has been rewritten using this capability. However, we believe that this is not a major threat, as the final version is what best reflects the intention of developers. Still, we may be missing some errors and problems appearing in the evolution, if they were corrected using history rewriting. This does not invalidate any of our findings, but may mean that more problems exist in practice. [3]

7.3 Threats to External Validity

External validity discusses how far the results are generalizable, or in other words how representative the sample of subjects and the circumstances of the experiment were, to be able to draw general conclusion. Do you expect the same results to be confirmed in somewhat modified conditions?

Thus external threats are primarily focusing with the relation of the experiment conditions to reality (in the industrial practice, etc).

To enlist external threats ask yourself whether there is any interaction of the treatment with the way you chosen the sample (for a different sample the treatment could have different effect)? and Whether the set up of the experiment interacts with the treatment? (in a different context, the treatment could have different effect)

7.4 Examples of Threats to External Validity

The following quote comes from a paper studying two feature models of operating systems (Linux and eCos) to understand the nature of models and variability modeling in this domain. The main challenge of that paper was that it only draws on a small number of subjects—only two home grown modeling languages, and one model in each of them:

> The main threat to the external validity of our findings is that they are based on two languages and two operating systems only. On the other hand, both are large independently developed real-world projects, with different objectives: Linux is a general purpose kernel and eCos is an en- tire specialized RTOS for embedded systems. We believe that other related domains, especially embedded RT such as automotive and avionic control software, will share many characteristics with the studied systems. Further, comparison to other feature modeling languages, shows that both are representative of the space of feature modeling. [1]

The above argument was a mitigation argument (why do we think we can generalize the results). The one in the example below, is a scoping argument: instead of mitigating we explain two what kinds of subjects the results can apply:

> The Linux development process requires adding features in a way that makes them immediately configurable. As a consequence, it not only enables immediate configurability, but also makes the entire code evolution feature-oriented. Arguably, such a process requires a significant amount of discipline and commitment that may be hard to find in other industrial projects.
>
> Not all projects assume closed and controlled variability model. Many projects are organized in plugin architectures, where variability is managed dynamically using extensions (for example Mozilla Firefox or Eclipse IDE). Our study does not provide any insight into evolution of variability in such projects. [3]

8 Related Work

We now discuss how to collect, study and describe related literature for your project.

8.1 How to Find Related Papers

First, seek in the literature you already know.

In the introductory lectures for this project cluster we have given an overview of literature (books, and some papers) on the main subjects of modeling and transformation.

Second, ask your supervisor

Ask your supervisor in the meeting for information about the important papers, or names of people, or relevant workshops, conferences and journals.

Third, search online repositories

Check past proceedings of relevant venues or previous and newer work of the same authors by checking their DBLP page. Simply google for “dblp John Smith” to find most papers by John Smith, and “dblp Universal Laziness 2011” to find papers published in “Universal Laziness” conference/workshop/journal in 2011.

DBLP (http://www.dblp.org/search/index.php) is the most important index of computing research literature. It tends to index trustworthy outlets. Besides very new, and very old work, it is *often* the case that papers not listed at DBLP are not resspectably published.

Furthermore, use scholar.google.com to search research literature by keywords and/or names. While DBLP is useful in establishing what papers to find, often it links to paid sources (you should have access to most of these sources from ITU's campus). Google will often be able o find a free version of a paper, once you know what you are searching for.

If you absolutely cannot find the paper online, when Google fails, when ITU subscription fails, when our library fails, then it is entirely OK to politely contact the authors of the paper, asking for a copy.

Fourth, Follow References

While reading papers, mark positions in related work that seem relevant to your work. Obtain the papers online and read these, too.

Seeking by related work, has one serious disadvantage: it only allows you to move back in the past. You end up finding older and older papers. It is however very likely, that the newest papers are relevant for your project. To identify these, you can use reverse searching in Google Scholar. Find a paper that is likely to be cited by papers in your work area, search for it in Google Scholar, and click on "Cited by" link, to find newer papers that cite it.

8.2 How To Select Papers To Read

With so many sources of relevant literature, you will often find out that there are dozens, if not hundreds papers, that appear interesting to read on your subject. This is why it is much more important to read the right papers, than to read a lot.

With this in mind I propose the following 4 step method for reading (or not!) a research paper:

1. You start with the title. If the title is not promising a relevant paper, then discard the idea — do not even print it!
2. Then continue with the abstract. After reading the abstract ask yourself: can this work benefit my work ? Can I claim that I improve on it ? Is this work addressing a similar problem? Same problem? Does it appear to address a problem better than what I am doing ? If you answer any of these questions positively, then the paper should survive on your reading list. Otherwise discard!
3. Read the introduction, related work, and conclusion (I really recommend to read the conclusion so early!). Ask yourself the same questions as before. Do you still think that this paper is relevant ? If so, it might be time to print it. Otherwise discard!

4. Hopefully only a few papers survive until this stage. Probably you need to read them all. Remember to assess them critically, by comparing evidence provided with the claims made (see previous lecture).

Resist the temptation to read (print) many interesting papers. Focus on relevant papers.

8.3 Reading

When you got to reading the paper entirely, it is crucial that you try to read through the entire paper relatively fast, deck to deck, without stopping. Since you already know the introduction, related work, and conclusion, the paper should be easier to read, than if you approached it entirely sequentially. Still, It is completely fine to skip over the paragraphs that you do not understand.

It is more important that you do the first reading reasonably fast. During (or after) that, you may want to visualize for yourself the main points, and structure, of the paper. For instance pencil down an outline or a mind-map for the paper. Critically assess the claims and the evidence provided (see last week).

Most regular papers, with exception of Journal papers and very mathematical works, should be possible to process within one hour in this manner (even half an hour should be enough for first quick reading of many software engineering papers).

After that, if there is need, you should read it again. Normally all the hard parts are much easier to understand at the second reading. It takes less time to read twice (first fast, then slow), than just to read once (only slow).

8.4 Citing

Your paper should contain some 15-30 references. You can easily assume that the last page in the LNCS format (and sometimes a bit more) will be used by references. About half of the references will be related work. The rest will be background info, references to tools you are using, origins of standard definitions, background information and alike.

Assess trustworthiness of each cited paper. Articles with lots of typos and language mistakes, badly organized, or not written clearly, are often also flawed conceptually. Workshop papers, often present only simple ideas, with no supporting data. This is the same for short conference papers (say up to 6-8 pages). Full size conference papers are more solid. Journal papers are typically long, they provide solid evidence and much context information.

It is also important to distinguish respected authors and respected venues. To check the former, you can see the publishing record of the author in DBLP (see above). Ask your advisor for help, to assess reputation of publication outlets.

Avoid discussing at length (and sometimes citing altogether) papers that you assess as not very trustworthy.

Wikipedia entries, industrial white papers, student term projects, and alike are usually not trustworthy sources to cite for research results. If you want to cite an idea, you should always strive to cite the primary source that published it, not a book or website that described it post factum (as opposed to inventing it). Wikipedia tends to cite primary sources for lots of factual information. So it might be a good starting point to find literature (but cite the primary sources after verification; do not to cite Wikipedia directly).

Avoid long quotations. A long quotation should really be an important one, and it should really be important that you use someone else's words, instead of your own. **Never ever quote without citing the source.**

It is also safest to avoid quoting figures. If you do this, investigate whether you have the right to do this, and always cite the original source.

If you are using LaTeX, then you will find BiBTeX entries for most papers in DBLP. This saves a lot of time, and gives your reference list a uniform look.

8.5 Writing The Related Work

It is important that you discuss predecessor works, and works with similar objectives in related work. It is not enough to mention them! It is not even enough to say how they are related. Quotes like "a related work is XXX", or "YYY solves a similar problem in a different way" are extremely weak. You have to explain precisely what are the advantages (and disadvantages, if any) of your approach. What improvement do you offer ? **For each of them.**

9 Conclusion

Summarize the most important outcomes of the paper. Make the sentences sound strong (punch!). It should contain the main points that you want the reader (and the examiner) to remember.

It is also customary to include a paragraph about future work: what would, or will, you do approach as the next research step after these results.

10 Title and Author List

The title should be informative and short. Space is always precious, so it should ideally fit on one or two lines. Shorter title is also easier for other authors to reference within space limits (and normally you want to be referenced). It is fine to have a catchy, almost newspaper style, title, but remember that this should not fool search engines. Most people find papers using Google.

There are two dominant schemes in author ordering:

- Alphabetical by last name; more common in theoretical papers
- By degree of contribution of each author: dominating in software engineering.

It is also traditional, that if you co-author a published work with a faculty member, for instance your supervisor, then the ordering by contribution is only applied to students co-authoring the work, while professors are listed in the end of the author list.

Regardless the scheme, always the first author gets most visibility, and becomes the reference name for the paper, eventually. So if you feel that you contributed significantly more than others, you should insist on ordering by contribution.

For a project report or a joint thesis, I recommend alphabetical, to avoid group conflicts. If you choose to order authors by contribution, then please take this discussion early in the group, to avoid conflicts in the last day.

Individuals, who only contributed to a paper marginally, should not be listed in the list of authors, but acknowledged. This includes all people who did not contribute to the ideas and the story, but for example only implemented some tool under direction of others. Also people who contributed marginally, for example by giving feedback to drafts, should not be authors, but should be acknowledged. Acknowledgments are listed in the very end, under conclusion in papers, and early in a preface or introduction in theses or books.

11 Format

Plase use maximum 15 pages of LNCS format. Springer provides style files at:

- LaTeX: ftp://ftp.springer.de/pub/tex/latex/llncs/latex2e/llncs2e.zip
- Microsoft Word: http://www.springer.com/cda/content/document/cda_downloaddocument/CSProceedings_AuthorTools_Word_2003.zip?SGWID=0-0-45-1124637-0

A claerly marked appendix can be added on top of the 15 pages. You cannot expect however that the appendix will be read, or assessed properly.

12 Homework due latest on November 7th

- Agree in the group what will be the experiment in your project (if any)
- Designate the person to design the experiment (this person may need to read up about it).
- Sketch (=**write down**) the experiment design and a short discussion of threats to validity.
- Discuss it with your supervisor next week.

Usually one can identify 2–4 main internal threats and the same amount of external threats that should be described. The size of a typical threats to validity section in the LNCS format would probably not exceed one page.

13 Homework due latest on November 14th

- Identify 5-10 (for now) papers that are related work for your own work.
- Show the list to your supervisor
- Explain how it is related, whether you improve on them, or whether they have similar objectives.
- If you are able, already draft the related work section based on that, now (will require editing later)

14 Later This Semester

Process from now on

- **Apr 3:** experiment designed
- **Apr 10:** threats to validity
- **Apr 17:** related work
- **Apr 23:** evaluation section written
- **Apr 28:** at 23.59 a complete draft for final review
- **May 5:** reviews due
- **May 15:** final deadline (to study admin)

References

[1] Thorsten Berger, Steven She, Rafael Lotufo, Andrzej Wasowski, and Krzysztof Czarnecki. Variability modeling in the real: a perspective from the operating systems domain. In Charles Pecheur, Jamie Andrews, and Elisabetta Di Nitto, editors, *ASE*, pages 73–82. ACM, 2010.

[2] Ralph E. Johnson, Kent Beck, Grady Booch, William R. Cook, Richard P. Gabriel, and Rebecca Wirfs-Brock. How to get a paper accepted at oopsla (panel). In *OOPSLA*, pages 429–436, 1993.

[3] Rafael Lotufo, Steven She, Thorsten Berger, Krzysztof Czarnecki, and Andrzej Wasowski. Evolution of the linux kernel variability model. In Jan Bosch and Jaejoon Lee, editors, *SPLC*, volume 6287 of *Lecture Notes in Computer Science*, pages 136–150. Springer, 2010.

[4] Mary Shaw. Writing good software engineering research paper. In *ICSE*, pages 726–737. IEEE Computer Society, 2003.

[5] Claes Wohlin, Per Runeson, Martin Host, Magnus C. Ohlsson, Bjorn Regnell, and Anders Wesslen. *Experimentation in Software Engineering*. Kluwer Academic Publishers, 2000.

[6] Justin Zobel. *Writing for Computer Science*. Springer Verlag, 2004.