Scientific Project: Databases for Multi-dimensional Data, Genomics & modern Hardware

Academic Writing I

David Broneske, Sebastian Dorok, Andreas Meister, Veit Köppen, Martin Schäler, and Gunter Saake
WS 2014/2015

© Christian Kästner, Carnegie Mellon University, USA
http://www.cs.cmu.edu/~ckaestne/
WHY TO WRITE A PAPER

• Communicate new findings
  • Publication = ultimate result of scientific research
  • Research is never finished until it's published

• To let the community know about your work
  • Recognition
  • Contacts, fruitful collaborations

• Get feedback from peers
  • External, independent, frank (anonymous)
IDEAS TO PAPER

Encode a complex web of ideas

... as a linear stream of text.

HOW?
DIFFERENCE: PAPER VS. RESEARCH

Paper organization ≠ Research process
AGENDA

• Structuring a Paper
• Getting Started
  • Organization Issues
  • Writing
  • Problems & Solutions
• Bibliography
CRITERIA

• Significance
  • Motivate why the research is important or useful. Explain what problem it addresses

• Clarity
  • Organize the paper well and write clearly. Make sure you support your claims

• Novelty
  • Extend the frontier of knowledge. Explicitly relate your research to previous work

• Correctness
  • Critically evaluate and support your claims with proofs, an implementation, examples, or experiments.

Source: William Cook: Academic Writing
ANATOMY OF A PAPER

- Title
- Abstract
- Introduction
- (Background / Related Work)
- (Problem Statement)
- Body
- Evaluation
- Discussion
- Related Work
- Conclusion + Future Work
- References
ABSTRACT

• Very brief summary of the paper
• Why is this work important, what was the motivation?
• Main contents, main results
• What is the contribution?
• Typically one of the last things to write

• => Is this paper relevant for the reader (and conference)?
INTRODUCTION

• What is the general problem? Why is it important?
• What is the specific problem? Why should the reader care?
• How is it different from prior work?
• What was the motivation for this work?
• What are the objectives/contributions? How is it new?
• What are the main results?
• What is the general approach/outline?
• Keep it short (approx. 1 column)
BACKGROUND (IF NECESSARY)

• What is the necessary background to understand this work?
• In scientific papers usually very short.
• Know your audience!
• Only background that is really necessary!
PROBLEM STATEMENT (IF NECESSARY)

• What is the specific problem? Why is it important?
• Example if necessary
• Sometimes necessary to tell the reader that there is a problem
THE CONTRIBUTION

• Main part of the paper
• Describes the own approach, the innovation
• Readable, verifiable! Examples where necessary!
EVALUATION / PROOF

• Evaluation critical
• What is the evaluation criteria?
• Case studies? Empirical studies?
• Does your innovation scale up? Does it solve real problems?
• Report experience
• Readable, verifiable! Can be assessed and replayed
• Separate data from interpretation
DISCUSSION (IF APPROPRIATE)

• Interpret results
• Advantages and Disadvantages
• (Comparison to related approaches)
• Threats to validity
RELATED WORK

• What are others doing?
• How does this differ from your work? (is your approach better? are there trade-offs? synergies?)
• Also discuss the relationship to YOUR prior work

• Claims of contribution are more convincing in the context of related work

• Common reviewer comments:
  • “The paper omits important related work”
  • “The authors describe the related work but don’t compare their work”
CONCLUSION AND FUTURE WORK

• Summary
• Results, what has been achieved
• What's missing? New research questions?
• Bigger context, long-term goals?

• Clarify the contribution with respect to the promises in abstract, introduction, and evaluation
REFERENCES

- Give credits to previous and contextual work
- Reference quotes, claims, previous results

- Only relevant, up-to-date references
- Prefer original source over secondary literature
- Prefer journal to conference to workshop to technical report to web pages

- Do not cite common knowledge (e.g., binary tree, propositional formula)
GETTING STARTED
WRITING IS WORK

• Few people enjoy to write and revise
• Writing is part of a profession

• Academic writing != fiction (inspiration, creativity, art)
• Writing to convey information
• Clarity instead of artistic prose

• -> Learn and practice
• -> Welcome feedback and criticism
WHY LEARN TO WRITE WELL

- Poorly written paper:
  - Ambiguity leads to misunderstanding
  - Omissions frustrate
  - Obscurity makes it difficult to reconstruct authors intentions
  - -> poor reviews, rejections
  - -> frustrated students
  - -> little impact
- Difficult to understand structure -> less focus on the content
- Even the best contribution is not convincing when it is difficult to understand
- Lazy presentation -> impression of unimportant work
GETTING STARTED

- Just write
- Make an outline or slides
  - Discuss this outline with your peers/supervisors

- Make a schedule and stick to it
- No excuses

- Write first, revise later
EXCUSES

- I can’t find time to write (I would write more if I had the time)
  - Schedule a time, commit to it!
- I need to do more analysis first / read more papers first
  - Do it in your scheduled time! Measure progress.
- I need new computer/printer/software/…
  - …
- Waiting till I feel like it / waiting for inspiration
  - Technical writing is work
  - Even novelists/poets reject notion of inspiration
- Writers block
  - Does not exist for technical writing
PLANNING
SCHEDULED WRITING

Abstinent (no non-emergency writing)

Spontaneous (50 sessions, when inspired)

Contingency Management (50 sessions, forced)

Boice 1990
MOTIVATIONAL TOOLS

• Setting goals
  • Overall goals, project goals
  • Plan deadlines
  • Concrete goal for each day (writing first three paragraphs of discussion section, write at least 200 words, revise Section 3, reconcile reference list, reread reviewers comments, …)

• Set priorities
  • Important vs Urgent

• Monitor progress
  • Simple table: date, project, #words, goals met
A paper is never "finished"

Improve by rewriting

Incrementally improve paper
FIRST STEPS

- Make an outline or make a presentation
- Write first version, revise later
\section{Introduction}

\$\text{SPL}$ introduction. Development of many variants in parallel, generation-compiled.

Many variants, testing etc $\rightarrow$ novel approaches needed

Preprocessor currently common, discussion about alternative implementations, \(1\)
whether long-term as well, tradeoffs, benefits, not discussed here

Type system for entire product lines (all variants are well typed), detection

Search for a simple solution, backward compatible, tool support, practical, so

Formalization for java subset, proof with coq, implementation for full java as
in several \$\text{SPLs}$ by others

Own and other prior work

Summary contributions
\section{Introduction}

SPL introduction. Development of many variants in parallel, generation-compile. A \textit{software product line (SPL)} is an efficient means to create a family of domain-specific \texttt{architectureBook, spleBook}. Instead of implementing each program from scratch, modeling a domain with features (increments in functionality relevant for stakeholders) from some assets that are common to the SPL (\texttt{foda, architectureBook, spleBook}), we can generate different variants, tailored to specific usage cases. This approach between the phases of SPL implementation (in which all variants are developed independently) and execution.

Many variants, testing etc. -> novel approaches needed. While the flexibility of SPLs to generate different tailored variants is an important strength (\texttt{architectureBook, spleBook}), it comes at a price of increased complexity. Developers implement virtually millions of variants in parallel. Testing SPLs in isolation means that each single product must be tested but potentially millions of different variants, even in which a certain feature or feature combination is selected (\texttt{spleBook, TF}), variants are never or rarely generated (e.g., only late after initial development). Potential errors might go undetected for a long time, until they are detected during testing or, compiling, and running all variants is not feasible for most SPLs. Therefore, novel approaches are needed that check the entire SPL itself instead of each individual variant.

Preprocessor currently common, discussion about alternative implementations, \texttt{\#} whether long-term as well, tradeoffs, benefits, not discussed here.

Type system for entire product lines (all variants are well typed), detection
TYPICAL PROBLEMS

• Missing motivation (why is it important?)
• Unclear goal, unclear contribution
• Missing reasoning (“that’s the way I did it”)
• Dead-end discussions, unused background
• Unjustified claims
• Missing cohesion
• Bigger picture missing (just details)
• Missing conclusions or results
• Jargon, background missing
• Related work missing
LINE OF THOUGHTS & COHESION (ROTER FADEN)

• Maintain cohesive line of thoughts
• Split text into paragraphs
  • Connect paragraphs
  • Do not jump between topics
• One thought per paragraph
  • Write topic sentence (e.g., first sentence or margin notes, \marginpar)
• Remove unnecessary information
Software product lines promise several benefits compared to individual development [Bass et al., 1998; Pohl et al., 2005]: Due to co-development and systematic reuse, software products can be produced faster, with lower costs, and higher quality. A decreased time to market allows companies to adapt to changed markets and to move into new markets quickly. Especially in embedded systems, in which resources are scarce and hardware is heterogeneous, efficient variants can be tailored to a specific device or use case [Beuche et al., 2004; Tešanović et al., 2004; Pohl et al., 2005; Rosenmüller et al., 2009]. There are many companies that report significant benefits from software product lines. For example, Bass et al. [1998] summarize that, with software product lines, Nokia can produce 30 instead of previously 4 phone models per year; Cummins, Inc. reduced development time for a software for a new diesel engine from one year to one week; Motorola observed a 400% increase in productivity; and so on.
MORE TOPIC SENTENCES

We decided to provide a formalization and proof for both properties, after an initial implementation of our type system for Java. We soon found that our implementation was incomplete: We could not give a guarantee and sometimes generated ill-typed variants because we forgot some

FJ is a minimal functional subset of the Java language for which typing and evaluation are specified formally and proved type-sound with the FJ calculus [8], [40]. It was designed to be compact; its syntax, type judgments and operational semantics fit on a single sheet of paper. FJ encodes features such as interfaces,

So far, we did not discuss the nature of feature annotations and the feature model. As illustrated in our examples in Section 3, we are interested in reachability conditions like the following sentence ‘whenever code fragment \(a\) is present, then also code fragment \(b\) is present’ based on their annotations and additional constraints of the feature model. (We use the metavariables \(a\) and \(b\) to refer to arbitrary annotatable code fragments.) Reachability is necessary, for example, to check whether a method invocation in code fragment \(a\) can always reference a method declaration in
COHERENCE

<table>
<thead>
<tr>
<th>Intro</th>
<th>Discussion</th>
</tr>
</thead>
<tbody>
<tr>
<td>The point (best)</td>
<td>...or here (ok)</td>
</tr>
</tbody>
</table>
COHERENCE ON A LARGE SCALE
SAY WHAT YOU SAY BEFORE YOU SAY IT

- Explain the structure of the text
- Pick up the readers, guide them, prepare them
- Connect chapters and sections
- Support readers in skimming the paper

7. IMPLEMENTATION & CASE STUDIES

In the previous sections, we have designed and formalized a product-line-aware type system. To demonstrate its practicality, we implemented it in our tool CIDE and performed a series of case studies to evaluate performance and whether we can actually find type errors in existing product lines.

7.1 Implementation

Benefits of AST representation

The AST representation has three main benefits: improved expressiveness, easier use, and opportunities for extensions.

First, we improve expressiveness, since we can classify more annotations as dis-
AVOID MERE DESCRIPTION

• Explain what you are doing and why

We implemented a type system in our tool CIDE and performed a series of case studies.

VS.

To demonstrate practicality, we implemented a type system in our tool CIDE and performed a series of case studies.
SELF CONTAINED

• You are an expert on the topic – your readers are probably not

• Provide all necessary background information for understanding your work
  • Be concise
  • Provide references for further details
  • A reference does not replace explaining necessary background

• Know your audience
STATING THE CONTRIBUTION

• Make contribution crystal clear
• Don’t be shy
• Be very specific: “we contribute”

The main innovation of this chapter is our revised type system for CFJ. The type system known from literature can be simplified due to redundant premises at the same typing rules. A smaller contribution is that we give some new and adapted examples of FJ programs and CFJ product lines.

Perspective, Goals, and Contributions. In this paper, we examine functional aspects in the light of AOR. Function evaluation imposes a fixed weaving order, but also a fixed refactoring order. That is, we cannot factor out aspect A.
STATING THE CONTRIBUTION
(EXAMPLE)

say that they are misused. To improve the situation, we make the following contributions:

- We analyze object-oriented modifiers used in FOP and identify several shortcomings that lead to a limited expressiveness of feature-oriented languages, undefined program behaviors, and inadvertent type errors.
- We explore the design space of feature-oriented access control mechanisms and propose three concrete access modifiers.
- We present an orthogonal access modifier model, which integrates common object-oriented modifiers with our novel feature-oriented modifiers.
- We offer an implementation of the proposed modifiers on top of the fully-fledged feature-oriented compiler FUJI.
- We analyze ten feature-oriented programs and demonstrate that there is a potential for feature-oriented modifiers in practical FOP.

Especially, the last two contributions are novel compared to an earlier version of the paper presented at FOSD’09 [11].
OVERCLAIMS

• Be careful with overclaims that you cannot prove
• Narrow it down to your actual contribution, be precise

Our approach provides reliable high-performance data access

Existing database systems are slow and do not scale
REFERENCING PUBLICATIONS

• Reference ideas and prior work

• Always reference used or adopted figures
  • e.g., “Figure 2: Feature model of Berkeley DB, adopted from [2]”
  • Copyright can be an issue

• NEVER copy and paste text from papers or websites
  • Paraphrase ideas
  • Also be careful when copying from yourself
  • More ethics on this later…
CITATION STYLE

• Direct quotations are not common, except for definitions
• Typically use quotation at the end of a sentence
  • „We formally extend Featherweight Java (FJ) – a Java subset proved type-sound using a concise calculus [41].“
  • „Without loss of generality, we focus on FODA-style feature models [12, 43], because ...“
  • „Parnas suggests dividing programs according to concerns instead of purely technical considerations [13].“
• Do not use reference as subject; avoid “see”
  • “[13] shows additional statistics” (bad)
  • “see [13] for additional statistics” (bad)
  • “In [13], Hu et al. show additional statistics” (borderline)
  • “Hu et al. present additional statistics [13]” (better)
CITING OWN WORK

• Make clear when referencing own work
  • “This problem was studied earlier, but in a less general setting [2,3,6].” (bad)
  • “We studied this problem earlier [2,3,6], but in a less general setting.” (better)
  • “In prior work, we studied this problem in a less general setting [2,3,6]” (better)
REFERENCE STYLE

• In papers
  • Typically numbered references are used [1], [2]
  • Page numbers omitted

• In a thesis
  • rather use abbreviations [ATG09] or better author-year style [Apel and Saake, 2006] (for Latex see package natbib)
  • Provide page numbers for books [S99, pp. 55-59]

• Different researchers prefer different styles. Ask advisers when writing a thesis. Check formatting guidelines of publishers.
FORMATTING BIBLIOGRAPHIES

• References must include
  • Name of authors
  • Title
  • Where published
    Journal Article: Journal & Volume & Edition & Pages
    Conference Paper: Conference & (Series and volume) & Pages & Publisher
    Book: Publisher
    Technical Report: Number & Department & University
  • Year
• ISBN, ISSN, DOI, location, date, editors and others are optional and usually not included (if you include them be consistent and include them for all references)
CLEAN YOUR BIBLIOGRAPHY

• An inconsistent/incomplete bibliography makes a bad impression, check consistency early on
• When importing bibtex entries, check for style and consistency
• Typical problems
  • Information missing (no publisher, no pages)
  • Inconsistent upper and lower case
    Classbox/j: Controlling the scope of change in java
    Aspect-Oriented Programming
  • Inconsistent names for conferences/journals, inconsistent abbrev.
    ICSE’08: Proceedings of the 30th International Conference on Software Engineering
    Proceedings International Conference on Software Engineering
TIP FOR Bibtex USERS: CONSTANTS FOR CONSISTENCY

@String{OOPSLA = "Proc.\ Int'l Conf.\ Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)"
}
@String{ICSE = "Proc.\ Int'l Conf.\ Software Engineering (ICSE)"
}
@String{ECOOP = "Proc.\ Europ.\ Conf.\ Object-Oriented Programming (ECOOP)"
}
@String{TSE = "IEEE Transactions on Software Engineering (TSE)"
}
@String{CACM = "Communications of the ACM"
}
@String{ViSPLE = "Proc.\ SPLC Workshop on Visualization in Software Product Line Engineering (ViSPLE)"
}
@String{LNCS = "Lecture Notes in Computer Science"
}
@String{GI = "Gesellschaft f\"ur Informatik (GI)"
}
@String{ACM = "ACM Press"
}
@String{Springer="Springer-Verlag"

@inproceedings{LBL:ICSE06,
    author = {Jia Liu and Don Batory and Christian Lengauer},
    title = {Feature Oriented Refactoring of Legacy Applications},
    booktitle = ICSE, publisher=ACM, address=ACMAddr, year = 2006,
    isbn = {1-59593-375-1}, pages = {112--121} }

Academic Writing I - Scientific project
EXAMPLES


NO PUBLISHER?

- Sometimes proceedings of workshops are published in technical reports by companies or universities

- When papers of a workshop are only published online, provide URL
REFERENCING URLs

• Don’t
• Consider using a footnote instead
• If you really must reference an URL, provide date of access
• If you can provide authors
• Reference specific version of wikis or other pages that keep a history
  • http://lampiro.googlecode.com/svn/!svn/bc/30/trunk/
QUESTIONS?

Thank you!

Thanks to Christian Kästner, Carnegie Mellon University, USA