Philipps Universität Marburg

Academic Writing

Paolo Giarrusso
(mostly based on material by Christian Kästner)

Basic premise of writing: information overload

- Readers don’t have time for you
- You can’t take readers attention or effort for granted
  - No point-last text
  - English vs. other traditions (German, Italian, East Asian?...)
- Hence, simplify the reader’s job.

Reading strategy

Papers:

- have **abstract**, **introduction** and **conclusions**
- those summarize message
- ⇒ read them to decide whether to read rest

Book chapters:

- a good book should follow the same idea
- for Code Complete, see for instance **Key Points** at the end.

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)

Why to write a survey papers

- Summarize existing findings
- Prepare for new research by summarizing the state of the art
- Get feedback from peers

When practicing:

- Test understanding
- Practice writing

Why to learn paper writing

All of the above, and:

- understand writing from others and learn to judge it
- learn to argue in a professional way

Encode a complex web of ideas

… as a linear stream of text.

HOW?

paper organization != research process

Criteria for research

- 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

Steps to writing a (term) paper

- “Ask a question worth answering” (*significance*).
- “Find an answer that you can support with good reasons” (*correctness*).
- “Find reliable evidence to support your reasons” (*correctness*).
- “Draft a report that makes a good case for your answer” (*correctness*).
- “Revise that draft until readers will think you met the previous goals” (*clarity*).

By *Turabian* (2007).

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

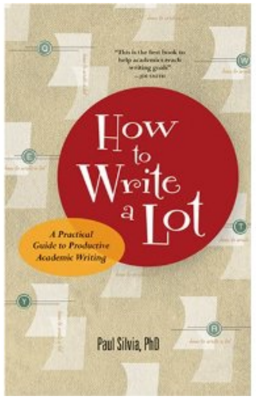
- Poorly written paper:
  - ambiguity leads to misunderstanding
  - omissions frustrate
  - obscurity makes it difficult to reconstruct authors intentions
  - → poor reviews, rejections
  - → frustrated students
  - → little impact
- Difficult to understand structure → less focus on the content
- Even 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 you peers/supervisers

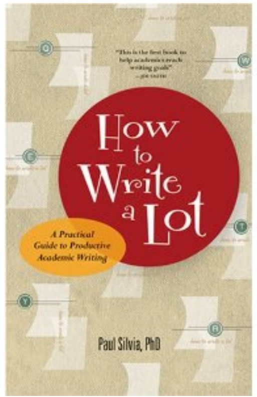
- Make a schedule and stick to it
- No excuses

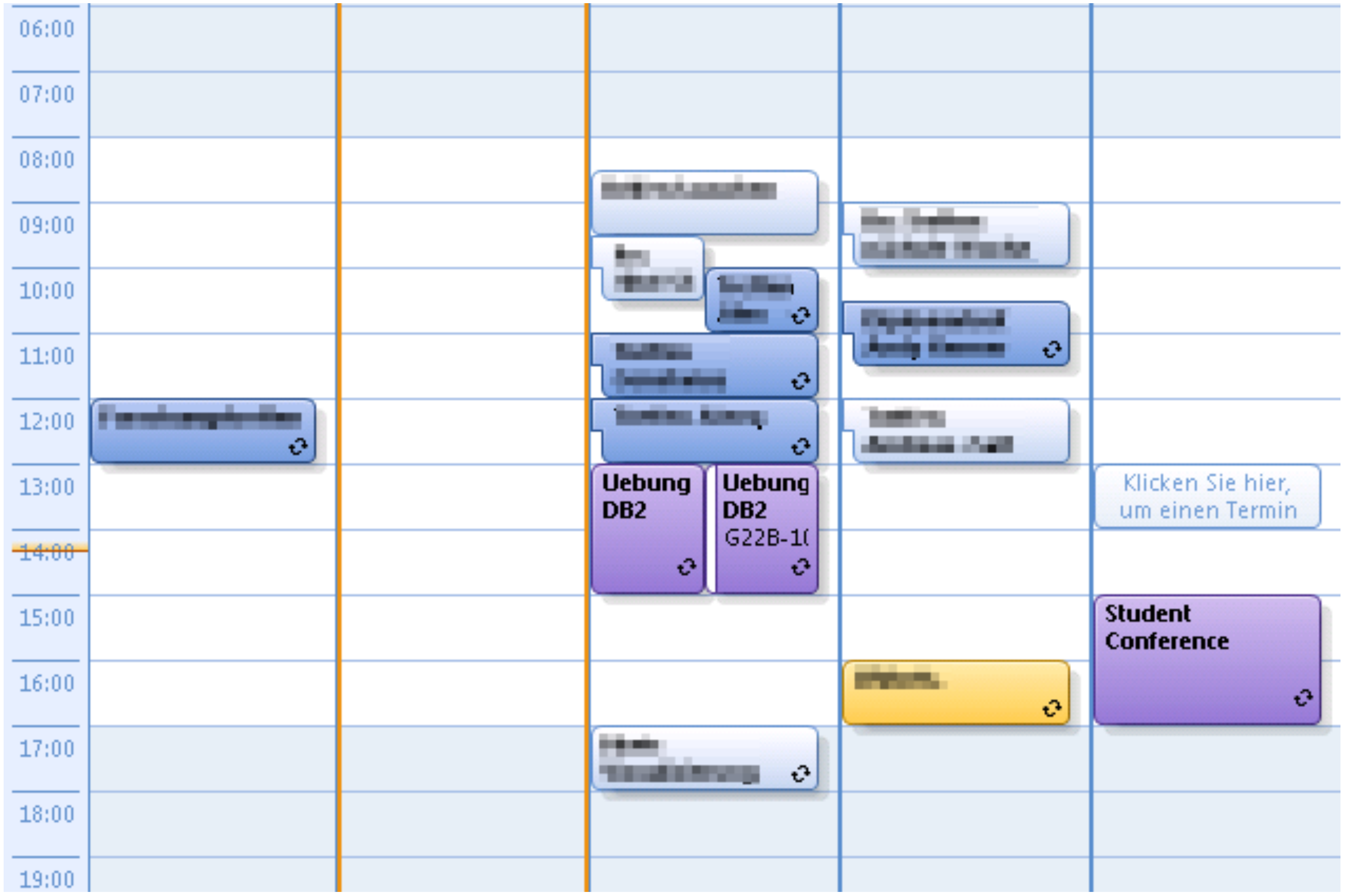
- Write first, revise later



Excuses

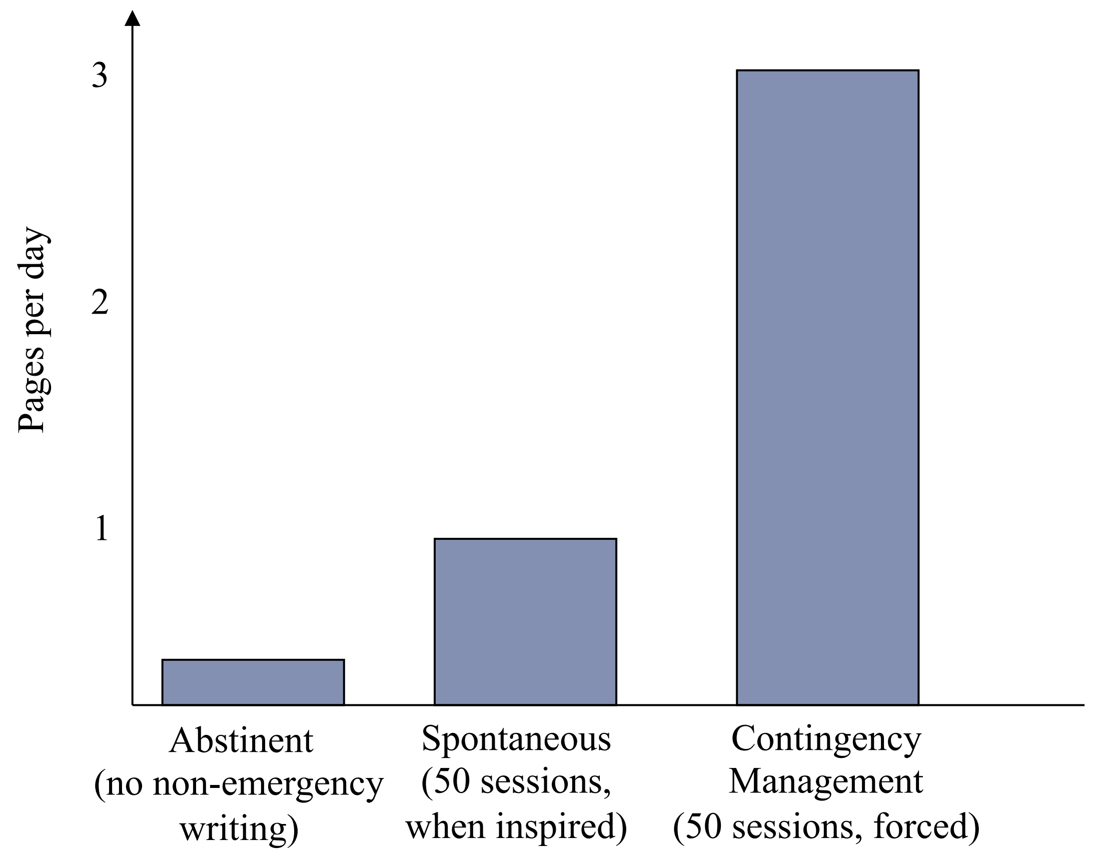
- I can't find time to write (I would write more if I had the time)
  - Schedule a time, commit to it!
- I need to do more analysis first / read more papers first
  - Do it in your scheduled time! Measure progress.
- I need new computer/printer/software/…
  - …
- Waiting till I feel like it / waiting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




06:00
07:00
08:00
09:00
10:00
11:00
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
Uebung DB2
Uebung DB2 G22B-1(
Klicken Sie hier, um einen Termin
Student Conference

Scheduled Writing

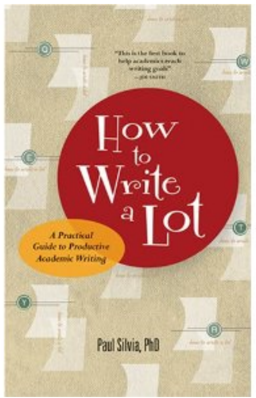
- Productivity gains:



Boice 1990

Motivational Tools

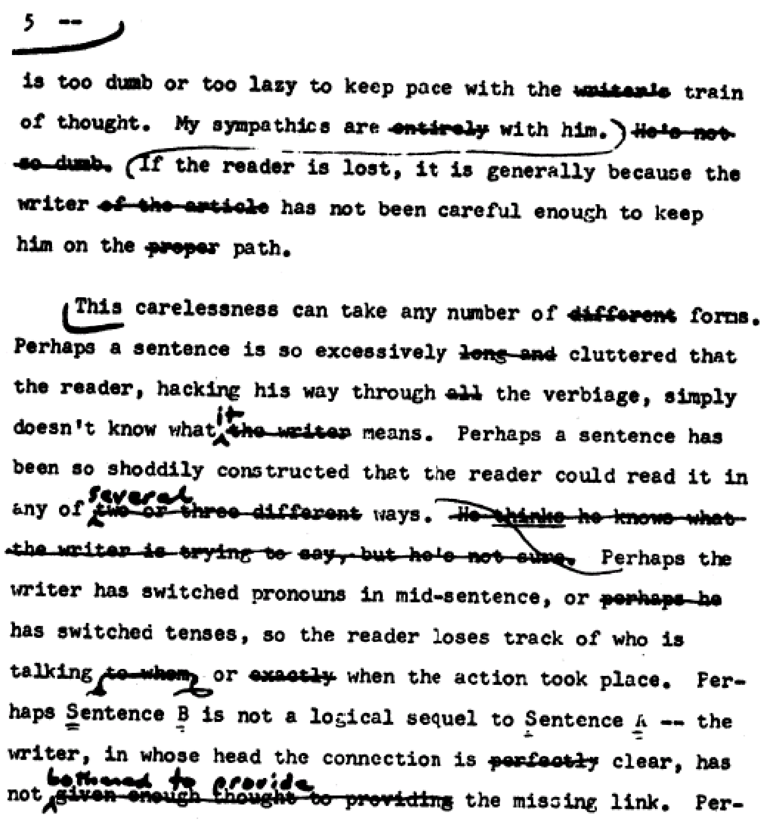
- Setting goals
  - Overall goals, project goals
  - Plan deadlines
  - Concrete goal for each day (writing first three paragraphs of discussion section, write at least 200 words, revise section 3, reconcile reference list, reread reviewers comments, …)
- Set priorities
  - Important vs Urgent
- Monitor progress
  - e.g. simple table: date, project, #words, goals met?



Rewrite

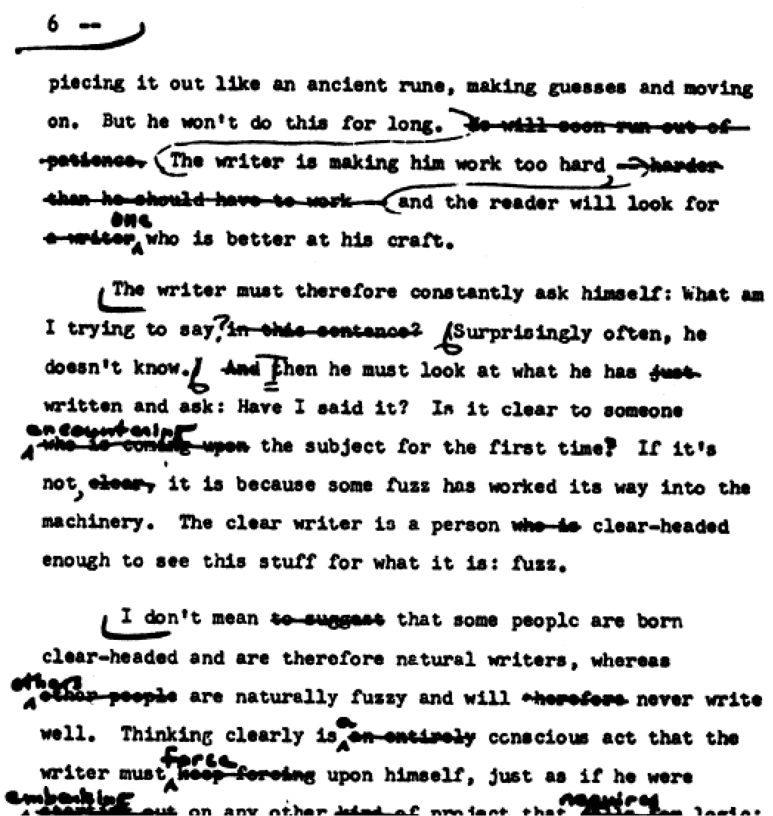
- A paper is never “finished”
- Improve by rewriting

- Incrementally improve paper


5 --

is too dumb or too lazy to keep pace with the ~~writer's~~ train of thought. My sympathies are ~~entirely~~ with him. ~~He's not so dumb.~~ If the reader is lost, it is generally because the writer ~~of the article~~ has not been careful enough to keep him on the ~~proper~~ path.

This carelessness can take any number of ~~different~~ forms. Perhaps a sentence is so excessively ~~long and~~ cluttered that the reader, hacking his way through ~~all~~ the verbiage, simply doesn't know what it ~~the writer~~ means. Perhaps a sentence has been so shoddily constructed that the reader could read it in any of several ~~two or three different~~ ways. ~~He thinks he knows what the writer is trying to say, but he's not sure.~~ Perhaps the writer has switched pronouns in mid-sentence, or ~~perhaps he~~ has switched tenses, so the reader loses track of who is talking ~~to whom~~ or ~~exactly~~ when the action took place. Perhaps Sentence B is not a logical sequel to Sentence A -- the writer, in whose head the connection is ~~perfectly~~ clear, has not bothered to provide ~~given enough thought to providing~~ the missing link. Per-


6 --

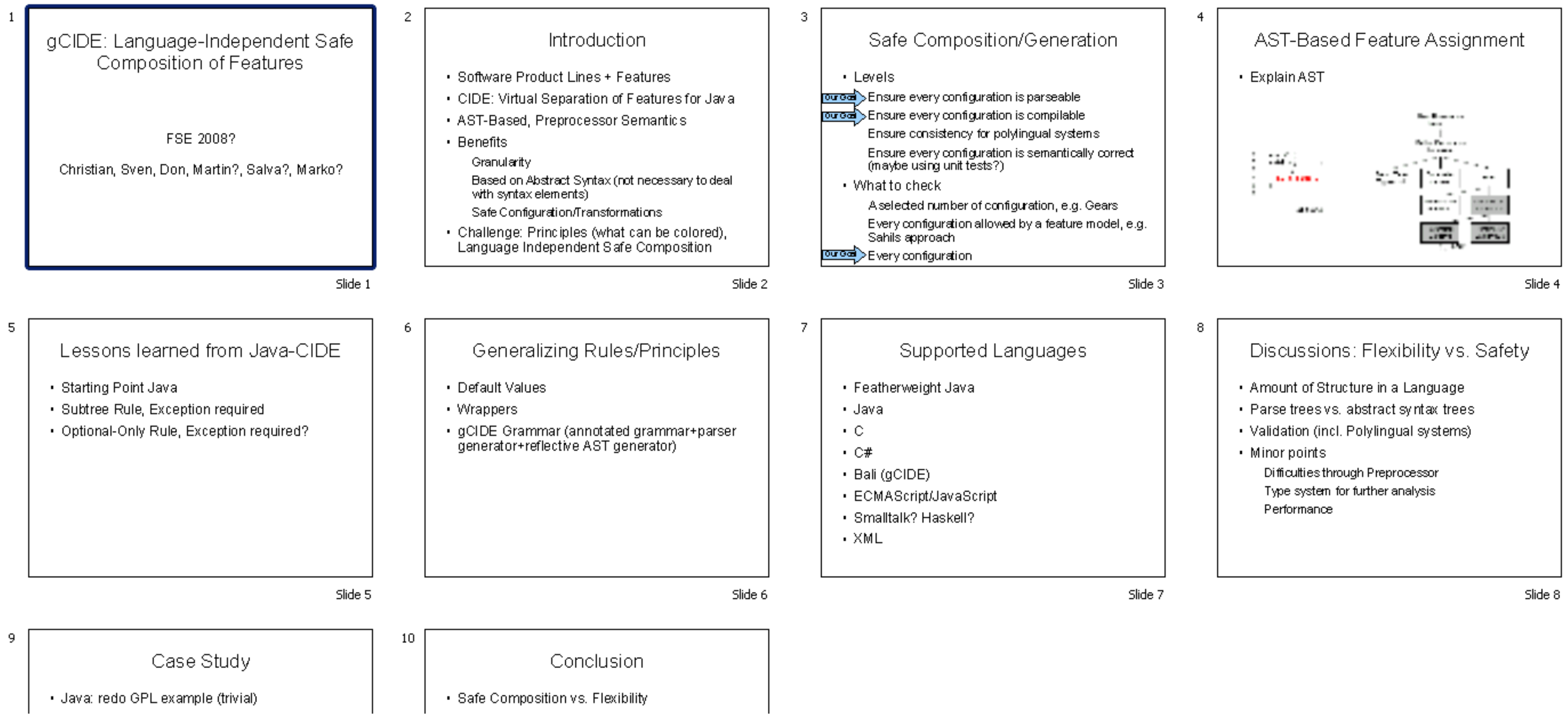
piecing it out like an ancient rune, making guesses and moving on. But he won't do this for long. ~~He will soon run out of patience.~~ The writer is making him work too hard, ~~harder than he should have to work~~ and the reader will look for one ~~a writer~~ who is better at his craft.

The writer must therefore constantly ask himself: What am I trying to say? ~~in this sentence?~~ Surprisingly often, he doesn't know. ~~And~~ Then he must look at what he has ~~just~~ written and ask: Have I said it? Is it clear to someone encountering ~~who is coming upon~~ the subject for the first time? If it's not, ~~clear,~~ it is because some fuzz has worked its way into the machinery. The clear writer is a person ~~who is~~ clear-headed enough to see this stuff for what it is: fuzz.

I don't mean ~~to suggest~~ that some people are born clear-headed and are therefore natural writers, whereas others ~~other people~~ are naturally fuzzy and will ~~therefore~~ never write well. Thinking clearly is a ~~an entirely~~ conscious act that the writer must force ~~keep forcing~~ upon himself, just as if he were embarking ~~[illegible]~~ out on any other ~~kind of~~ project that requires ~~[illegible]~~ logic:

First Steps

- Make an outline
- Or make a presentation
- Write first version, revise later



1

gCIDE: Language-Independent Safe Composition of Features

FSE 2008?

Christian, Sven, Don, Martin?, Salva?, Marko?

Slide 1

2

Introduction

- Software Product Lines + Features
- CIDE: Virtual Separation of Features for Java
- AST-Based, Preprocessor Semantics
- Benefits
  - Granularity
  - Based on Abstract Syntax (not necessary to deal with syntax elements)
  - Safe Configuration/Transformations
- Challenge: Principles (what can be colored), Language Independent Safe Composition

Slide 2

3

Safe Composition/Generation

- Levels
  - Our Goal Ensure every configuration is parseable
  - Our Goal Ensure every configuration is compilable
  - Ensure consistency for polylingual systems
  - Ensure every configuration is semantically correct (maybe using unit tests?)
- What to check
  - A selected number of configuration, e.g. Gears
  - Every configuration allowed by a feature model, e.g. Sahils approach
  - Our Goal Every configuration

Slide 3

4

AST-Based Feature Assignment

- Explain AST

Slide 4

5

Lessons learned from Java-CIDE

- Starting Point Java
- Subtree Rule, Exception required
- Optional-Only Rule, Exception required?

Slide 5

6

Generalizing Rules/Principles

- Default Values
- Wrappers
- gCIDE Grammar (annotated grammar+parser generator+reflective AST generator)

Slide 6

7

Supported Languages

- Featherweight Java
- Java
- C
- C#
- Bali (gCIDE)
- ECMAScript/JavaScript
- Smalltalk? Haskell?
- XML

Slide 7

8

Discussions: Flexibility vs. Safety

- Amount of Structure in a Language
- Parse trees vs. abstract syntax trees
- Validation (incl. Polylingual systems)
- Minor points
  - Difficulties through Preprocessor
  - Type system for further analysis
  - Performance

Slide 8

9

Case Study

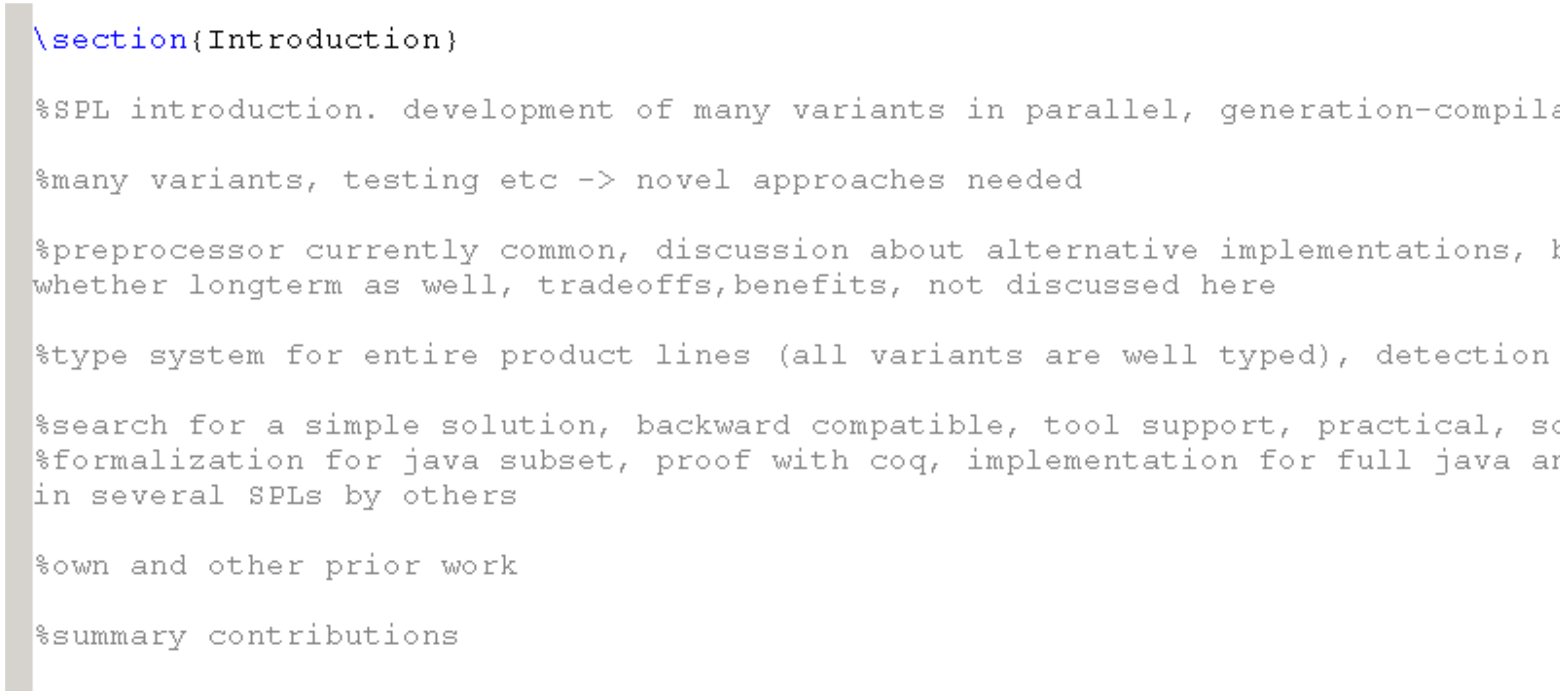
- Java: redo GPL example (trivial)

10

Conclusion

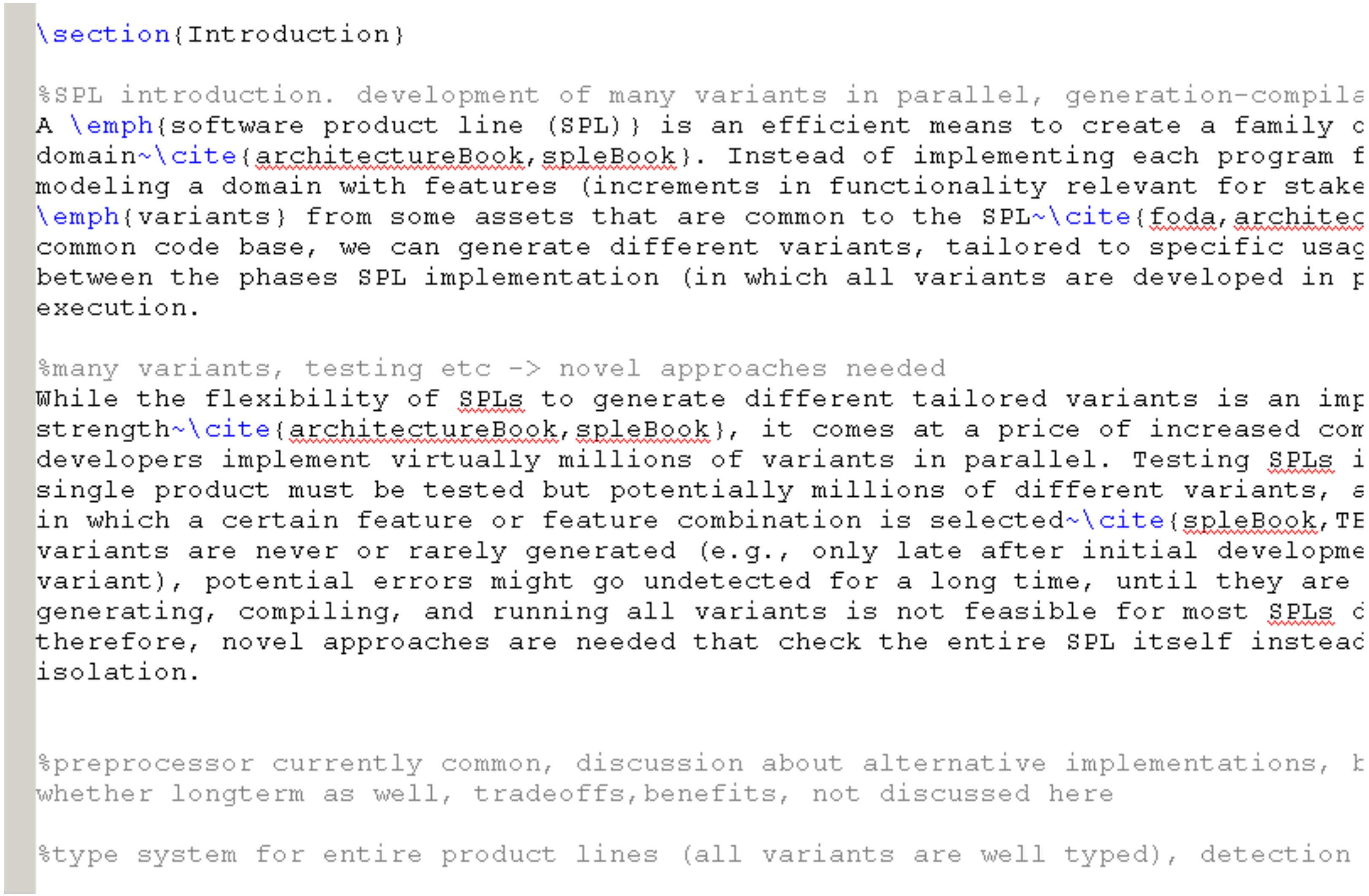
- Safe Composition vs. Flexibility

First Steps



```
\section{Introduction}

%SPL introduction. development of many variants in parallel, generation-compila

%many variants, testing etc -> novel approaches needed

%preprocessor currently common, discussion about alternative implementations, k
whether longterm 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, sc
%formalization for java subset, proof with coq, implementation for full java ar
in several SPLs by others

%own and other prior work

%summary contributions
```

First Steps



```
\section{Introduction}

%SPL introduction. development of many variants in parallel, generation-compila
A \emph{software product line (SPL)} is an efficient means to create a family c
domain~\cite{architectureBook,spleBook}. Instead of implementing each program f
modeling a domain with features (increments in functionality relevant for stake
\emph{variants} from some assets that are common to the SPL~\cite{foda,architec
common code base, we can generate different variants, tailored to specific usag
between the phases SPL implementation (in which all variants are developed in p
execution.

%many variants, testing etc -> novel approaches needed
While the flexibility of SPLs to generate different tailored variants is an imp
strength~\cite{architectureBook,spleBook}, it comes at a price of increased com
developers implement virtually millions of variants in parallel. Testing SPLs i
single product must be tested but potentially millions of different variants, a
in which a certain feature or feature combination is selected~\cite{spleBook,TE
variants are never or rarely generated (e.g., only late after initial developme
variant), potential errors might go undetected for a long time, until they are
generating, compiling, and running all variants is not feasible for most SPLs c
therefore, novel approaches are needed that check the entire SPL itself instead
isolation.


%preprocessor currently common, discussion about alternative implementations, k
whether longterm 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

Revising for Clarity: Sentences

Revising a sentence for clarity – Example 1

- Bad: “**Termination** *occurred* after 23 iterations”
- Good: “The **program** **terminated** after 23 iterations.”

Goal:

- make **Actor** explicit
- as *subject*

Revising a sentence for clarity – Example 2

- Bad: “**Determination** of policy *occurs* at the presidential level”
- Good: “The **President** **determines** policy”

Revising a sentence for clarity – Example 3

- Bad: “*There is* a **need** for further **study** of this program”
- Good: “The **engineering staff** **must study** this program further”

Revising a sentence for clarity – Key idea

- You're telling a story
- Figure out actions, and the agents doing them
- **Action = verb**
- **Agent = subject**
  - Agents sometimes might be abstract, if they're familiar to readers or the abstractions are critical.

Revising a sentence for clarity

Consequences (not fixed rules):

- Try to limit empty/weak verbs
  - Example: “**perform** typechecking” -> typecheck
  - “there is a <nominalization>”
- Limit passive (but see later)

Revising a sentence for clarity

Consequences (not fixed rules):

- Limit metadiscourse
  - Bad: “**It seems to us a plausible conjecture** that …”
  - Don’t give it the main verb, move it aside.
  - Better: “… in our opinion …”/”… according to our conjecture …”
  - If metadiscourse is important: “we conjecture”

Revising: Beyond Single Sentences

Revising sentences together: **cohesion**

- Go from old topics to new, avoid jumping between topics.
- This might require using passive.

Passive is fine for cohesion

- Good:
  "We thought we had a good agreement. Then we found out who killed it: The agreement **was broken by** the *partners*."

- Bad:
  "We thought we had a good agreement. Then we found out who killed it: The *partners* **broke** the agreement."

Repetition

- Good:
  "We thought we had a good **agreement**. Then we found out who killed it: The **agreement** was broken by the *partners*."

- Bad:
  "We thought we had a good **agreement**. Then we found out who killed it: The partners broke the **agreement**."

Repetition

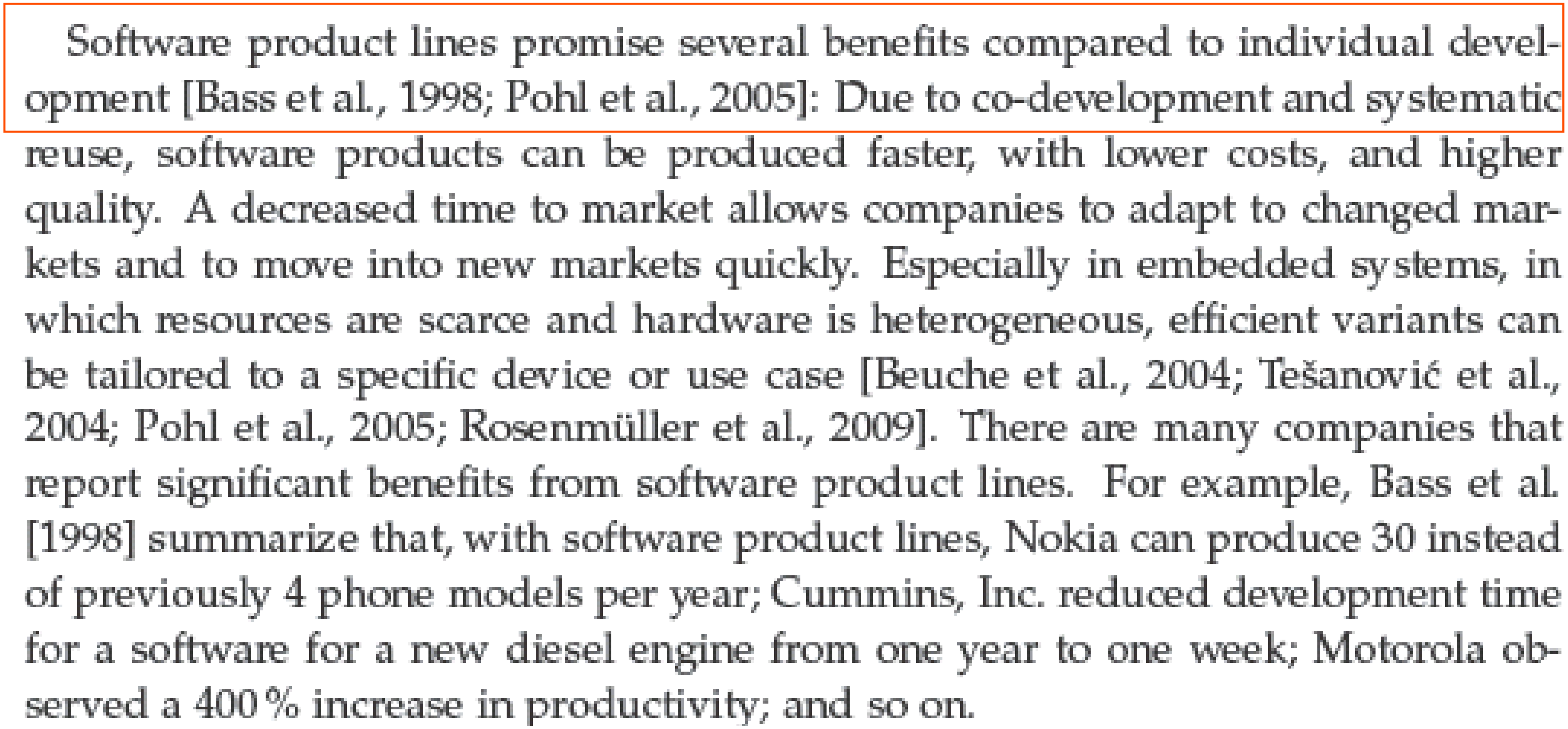
"Don't reuse the same word" is common advice, but has lots of downsides:

- synonyms for technical terms can be confusing (readers need to learn more terms than otherwise needed)
- pronouns can be used **if they are not ambiguous**

Line of Thoughts & Cohesion (Roter Faden)

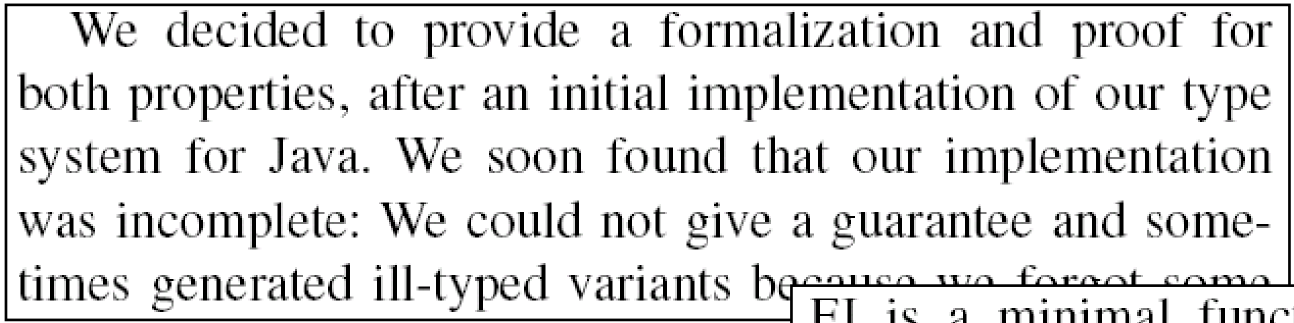
- Maintain cohesive line of thoughts
- Split text into paragraphs
  - connect paragraphs
  - do not jump between topics
- One thought per paragraph
  - Write topic sentence (e.g., first sentence or margin notes, \marginpar)
- Remove unnecessary information

Topic Sentence – Example

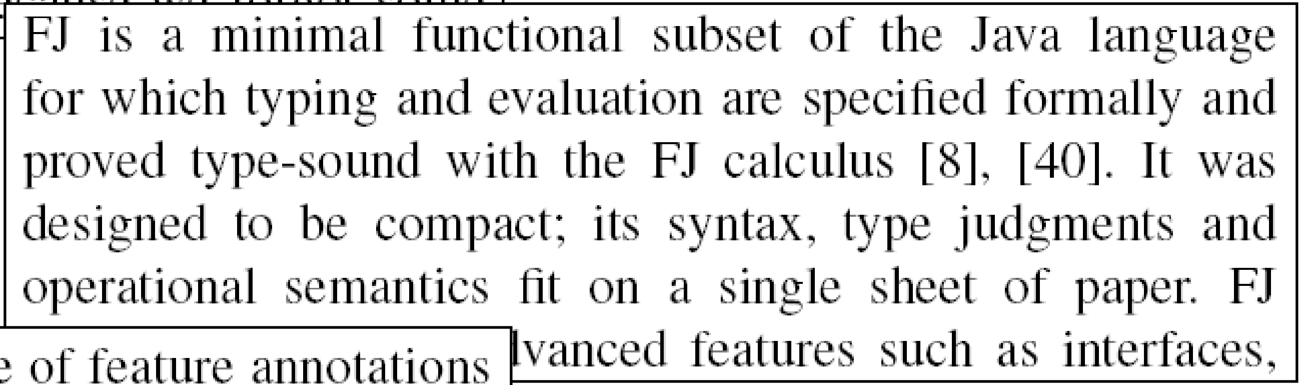


Software product lines promise several benefits compared to individual development [Bass et al., 1998; Pohl et al., 2005]: Due to co-development and systematic reuse, software products can be produced faster, with lower costs, and higher quality. A decreased time to market allows companies to adapt to 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

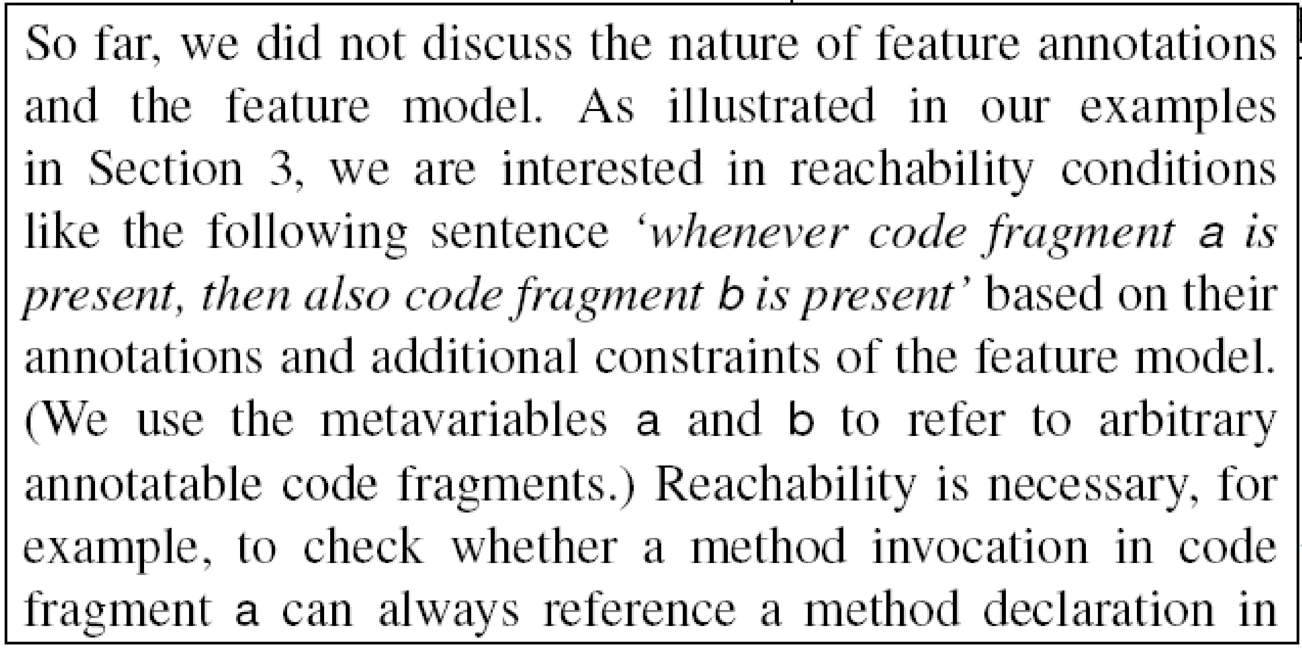


We decided to provide a formalization and proof for both properties, after an initial implementation of our type system for Java. We soon found that our implementation was incomplete: We could not give a guarantee and sometimes generated ill-typed variants be



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

lvanced features such as interfaces,

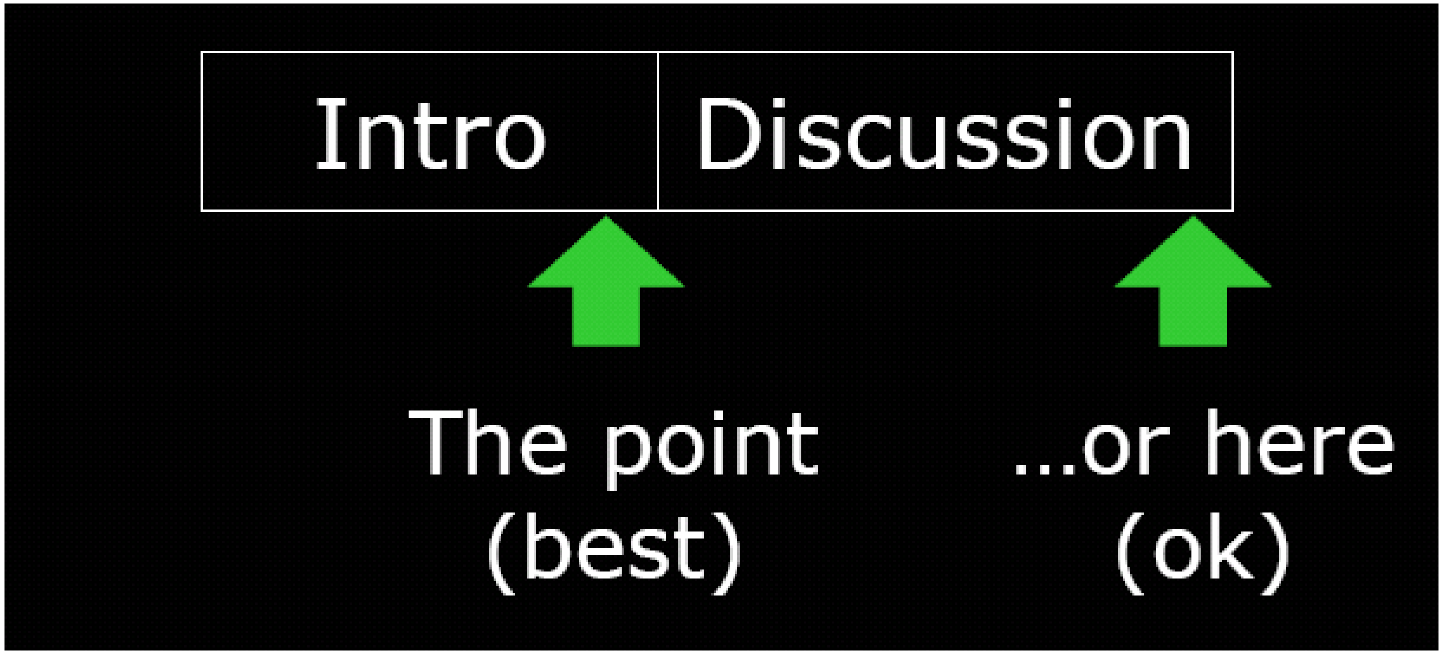


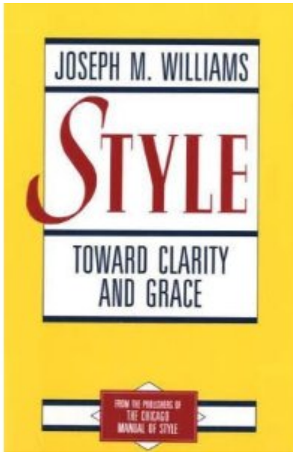
So far, we did not discuss the nature of feature annotations and the feature model. As illustrated in our examples in Section 3, we are interested in reachability conditions like the following sentence '*whenever code fragment* `a` *is present, then also code fragment* `b` *is present*' based on their 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

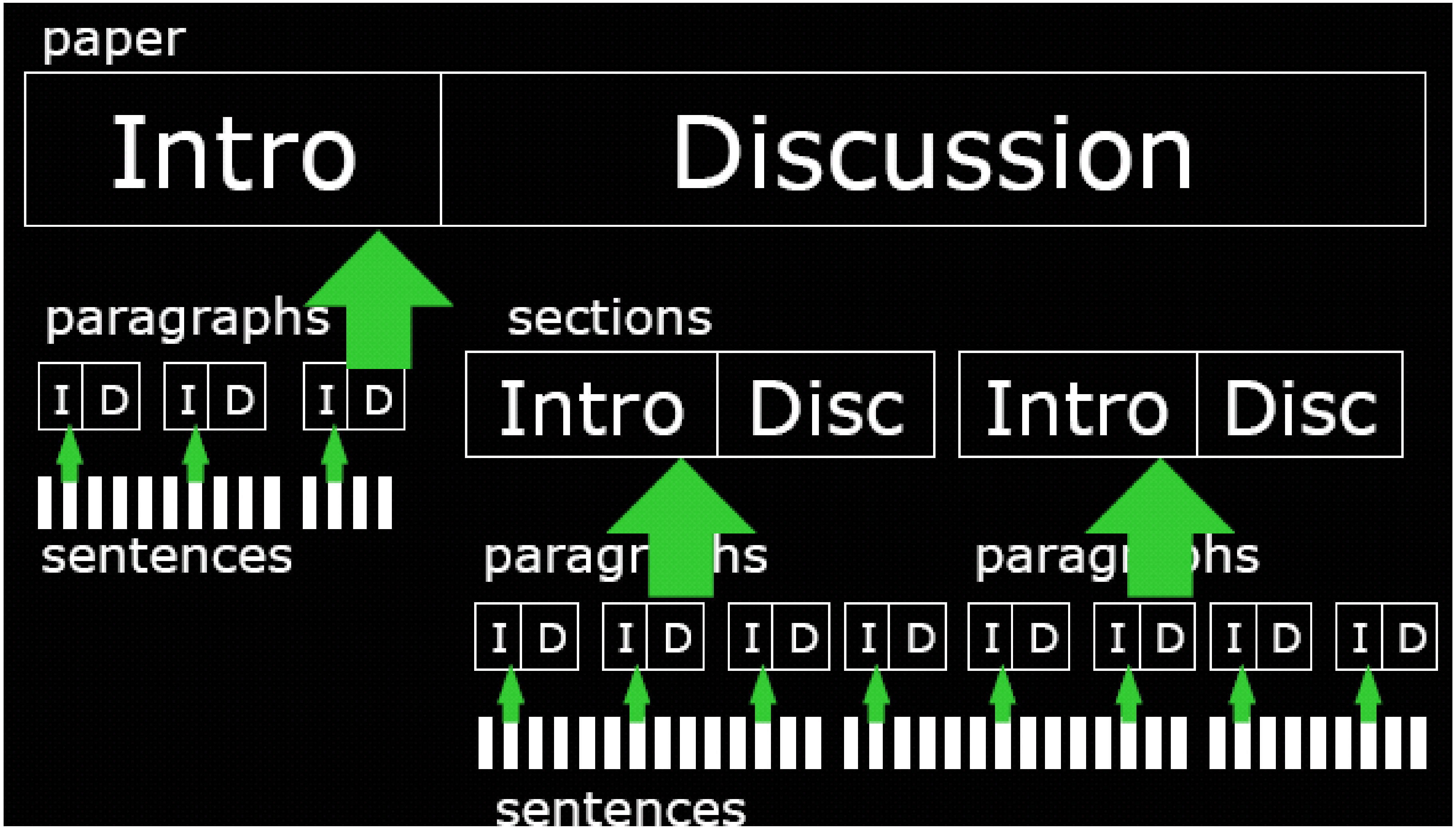
- Paper = Intro + Discussion
  - Best make your point in the intro, then elaborate.
- A paper is made by sections.
  - Each section should state its point at the beginning, then elaborate.
- A section is made by paragraphs.
  - Each paragraph should state its point at the beginning, then elaborate.

Coherence


Intro
Discussion
The point (best)
...or here (ok)

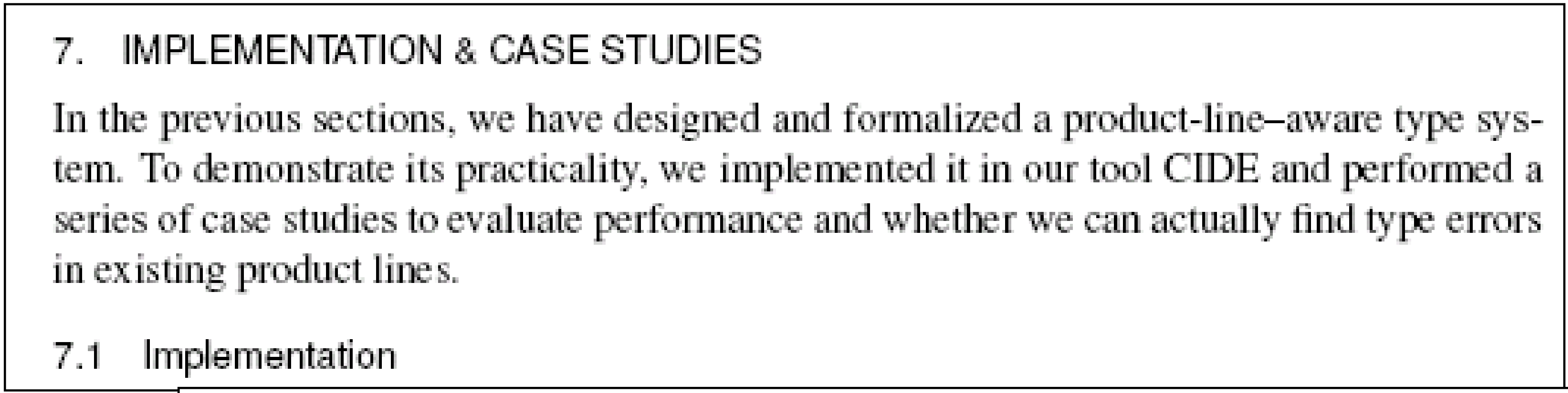

JOSEPH M. WILLIAMS
STYLE
TOWARD CLARITY AND GRACE

Coherence on a Large Scale



Say what you say before you say it

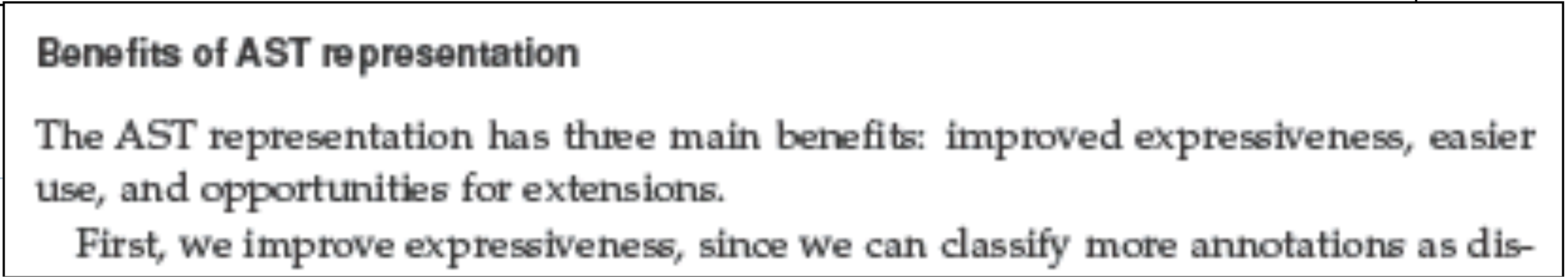
- Explain the structure of the text
- Pick up the readers, guide them, prepare them
- Connect chapters and sections
- Support readers in skimming the paper („Querlesen“)



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.

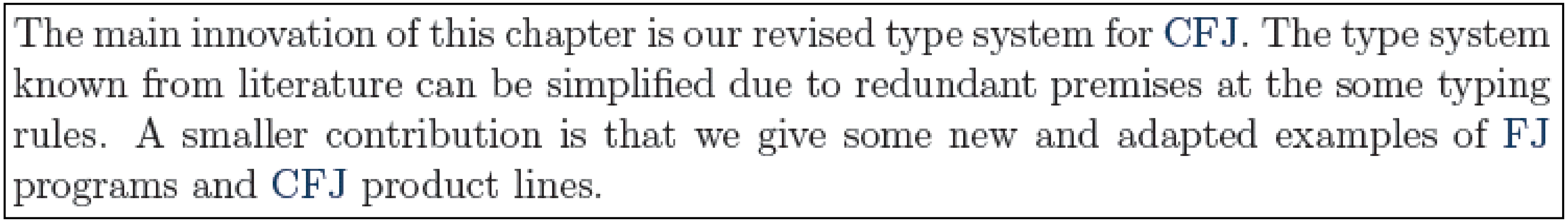
Self Contained

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

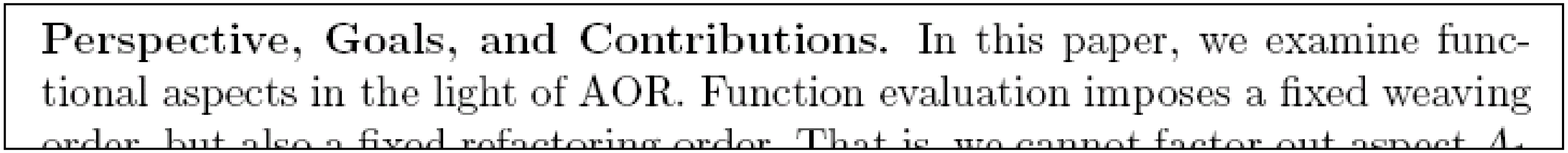
- Provide all necessary background information for understanding your work
  - Be concise
  - Provide references for further details
  - A reference does not replace explaining necessary background
- Know your audience

Stating the Contribution

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

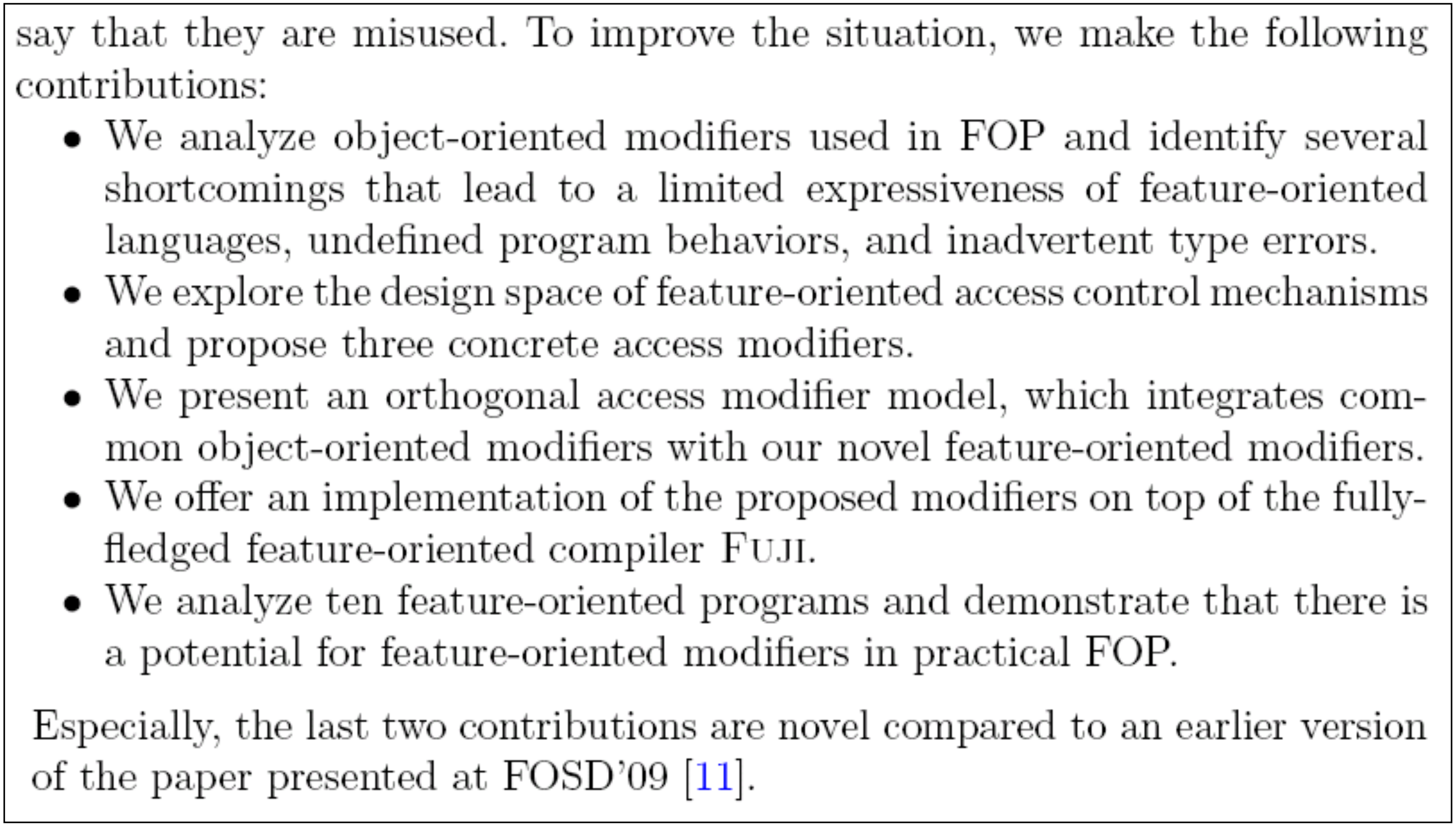


The main innovation of this chapter is our revised type system for CFJ. The type system known from literature can be simplified due to redundant premises at the some typing rules. A smaller contribution is that we give some new and adapted examples of FJ programs and CFJ product lines.



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

Bibliography

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. presented additional statistics [13]” (better)

Citing own work

- Make clear when referencing own work
  - “This problem was studied earlier, but in a less general setting [2,3,5].” (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.
    - Proc. Int'l Conf. Software Engineering (ICSE)
    - 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{\"u}r 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} }
```

Examples

- Rick Rabiser, Paul Grünbacher, and Deepak Dhungana. Supporting product derivation by adapting and augmenting variability models. In Proc. Int’l Software Product Line Conference (SPLC), pages 141–150, IEEE Computer Society, 2007.
- Christian Prehofer. Feature-oriented programming: A fresh look at objects. In Proc. Europ. Conf. Object-Oriented Programming (ECOOP), volume 1241 of Lecture Notes in Computer Science, pages 419–443, Springer-Verlag, 1997.
- Benjamin C. Pierce. Types and Programming Languages. MIT Press, 2002.
- David L. Parnas. Designing software for ease of extension and contraction. IEEE Transactions on Software Engineering (TSE), SE-5(2):128–138, 1979.

No Publisher?

- Sometimes proceedings of workshops are published in technical reports by companies or universities
  - Florian Heidenreich, Ilie ¸Savga, and ChristianWende. On controlled visualisations in software product line engineering. In Proc. SPLC Workshop on Visualization in Software Product Line Engineering (ViSPLE), pages 303–313, Lero, 2008a.
- When papers of a workshop are only published online, provide URL
  - Sean McDirmid and Martin Odersky. The Scala plugin for Eclipse. In Proc. ECOOP Workshop on Eclipse Technology eXchange (ETX), 2006. published online http://atlanmod.emn.fr/www/papers/eTX2006/.

Referencing URLs

- Don’t

- Consider using a footnote instead
- If you really must reference an URL, provide date of access
  - Eclipse Website, http://eclipse.org, accessed June 12, 2009
- If you can provide authors
  - LE BERRE, D., PARRAIN, A., ROUSSEL, O., AND SAIS, L. 2006. SAT4J: A satisfiability library for Java. http://www.sat4j.org.
- Reference specific version of wikis or other pages that keep a history
  - http://en.wikipedia.org/w/index.php?title=Bibliography&oldid=351449917
  - http://lampiro.googlecode.com/svn/!svn/bc/30/trunk/

For further reading

On Productivity/Procrastination:

- Paul J. Silvia. 2007. *How to Write a Lot: A Practical Guide to Productive Academic Writing.* American Psychological Association.

For further reading

On writing:

- Joseph M. Williams, Gregory G. Colomb. 1995. *Style: Toward Clarity and Grace.* University of Chicago Press.

Online summaries:

- http://www.cs.utexas.edu/~wcook/Courses/398T-F08/2008-writing.pdf
- http://explorationsofstyle.com/2011/03/02/verbs/