



Research Topics in SE

Zhendong Su

Objectives

- Learn to **present** technical work
- Learn to **understand** & **evaluate** research papers
- Learn several key **research** directions in the area

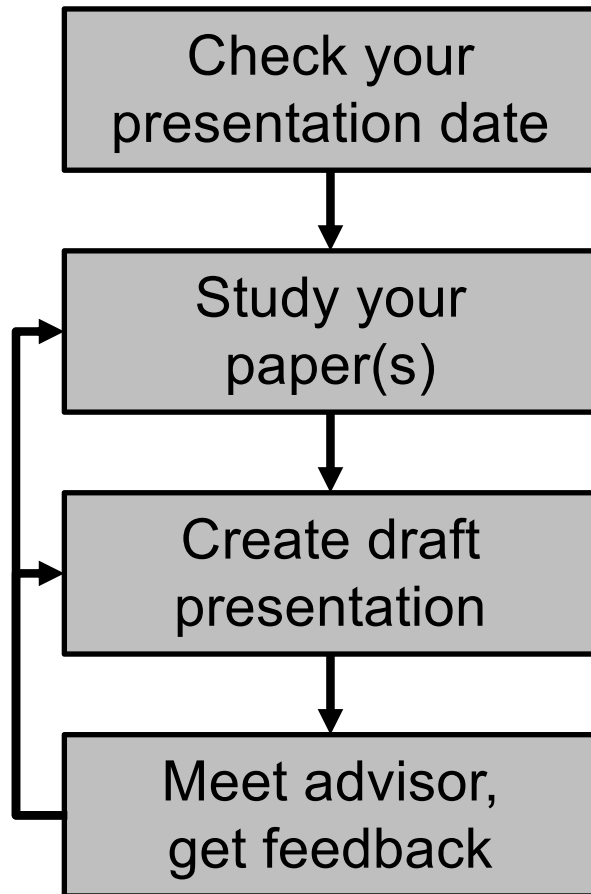
Objectives

- Learn to **present** technical work
- Learn to **understand** & **evaluate** research papers
- Learn several key **research** directions in the area
- Have a good deal of **fun** doing so

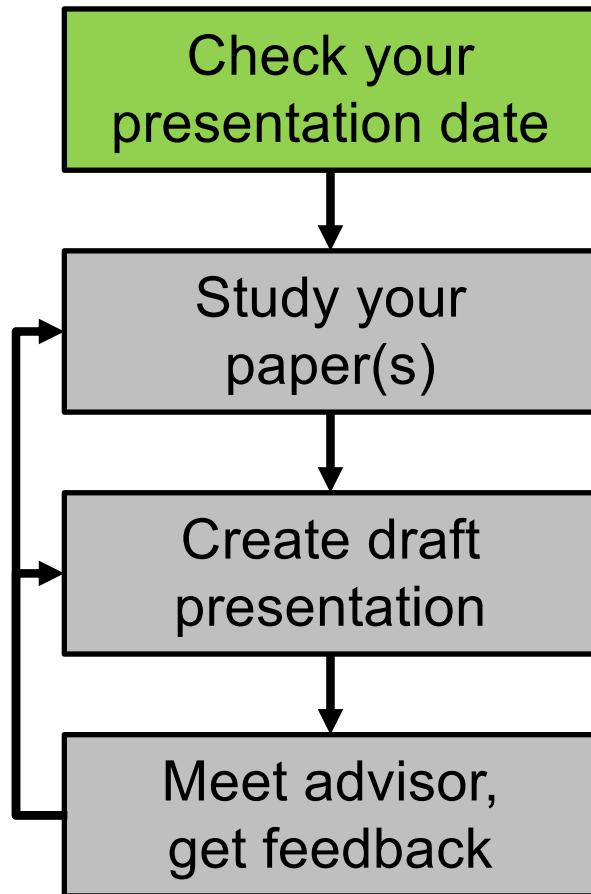
Paper Pool for Spring 2020

1. [Coq Coq Correct! Verification of Type Checking and Erasure for Coq, in Coq](#). POPL 2020.
2. [Executable formal semantics for the POSIX shell](#). POPL 2020.
3. [Incorrectness Logic](#). POPL 2020.
4. [Debugging Inputs](#). ICSE 2020.
5. [Quickly Generating Diverse Valid Test Inputs with Reinforcement Learning](#). ICSE 2020.
6. [Adversarial sample detection for deep neural network through model mutation testing](#). ICSE 2019.
7. [Fuzz Testing based Data Augmentation to Improve Robustness of Deep Neural Networks](#). ICSE 2020.
8. [Testing DNN Image Classifier for Confusion & Bias Errors](#). ICSE 2020.
9. [TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing](#). ICML 2019.
10. [Defending Against Physically Realizable Attacks on Image Classification](#). ICLR 2020.
11. [Chopped Symbolic Execution](#). ICSE 2018.
12. [Parser-Directed Fuzzing](#). PLDI 2019.
13. [Initialize Once, Start Fast: Application Initialization at Build Time](#). OOPSLA 2019.
14. [Excelint: Automatically Finding Spreadsheet Formula Errors](#). OOPSLA 2018.
15. [Memento mori: dynamic allocation-site-based optimizations](#). ISMM 2015.
16. [Defining the Undefinedness of C](#). PLDI 2015.
17. [Efficient Lock-Free Durable Sets](#). OOPSLA 2019.
18. [Mesh: Compacting Memory Management for C/C++ Applications](#). PLDI 2019.
19. [Virtual Machine Design for Parallel Dynamic Programming Languages](#). OOPSLA 2018.
20. [REPT: Reverse Debugging of Failures in Deployed Software](#). OSDI 2018.
21. [Automated clustering and program repair for introductory programming assignments](#). PLDI 2018.
22. [A Systematic Literature Review of Automated Feedback Generation for Programming Exercises](#). TOCE 2019.
23. [Rethinking Debugging as Productive Failure for CS Education](#). SIGCSE 2019.
24. [Does syntax highlighting help programming novices?](#). ESE 23 (2018).
25. [What distinguishes great software engineers?](#). ESE 25 (2020).
26. [Finding Crash-Consistency Bugs with Bounded Black-Box Crash Testing](#). OSDI 2018.
27. [HyDiff: Hybrid Differential Software Analysis](#). ICSE 2020.
28. [Causal Testing: Understanding Defects' Root Causes](#). ICSE 2020.
29. [Synthesizing Database Programs for Schema Refactoring](#). PLDI 2019.
30. [Coverage Guided, Property Based Testing](#). OOPSLA 2019.
31. [SLING: Using Dynamic Analysis to Infer Program Invariants in Separation Logic](#). PLDI 2019.
32. [HOPPITY: LEARNING GRAPH TRANSFORMATIONS TO DETECT AND FIX BUGS IN PROGRAMS](#). ICLR 2020.
33. [Scalable Taint Specification Inference with Big Code](#). PLDI 2019.
34. [code2vec: Learning Distributed Representations of Code](#). POPL 2019.
35. [Code vectors: understanding programs through embedded abstracted symbolic traces](#). ESEC/FSE 2018.

Preparing a Talk

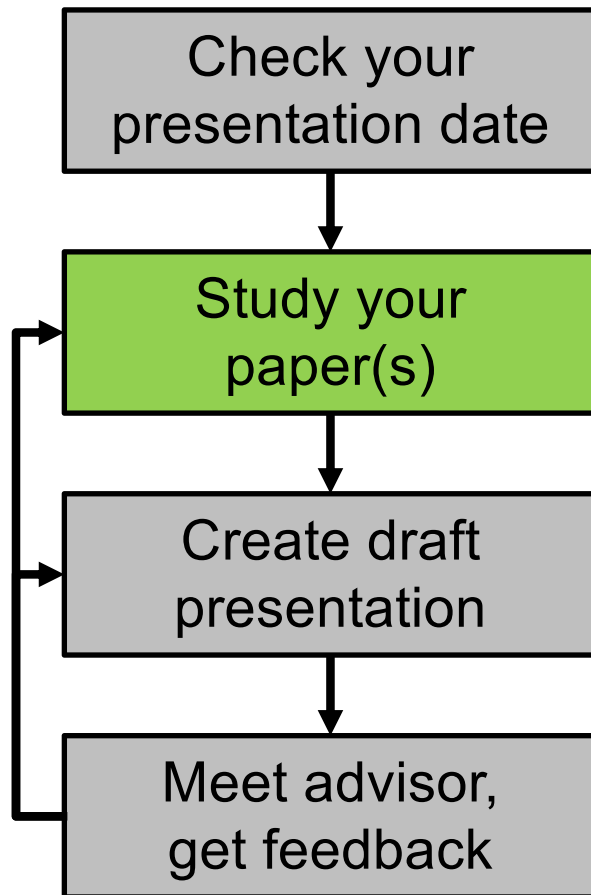


Preparing a Talk: **Start Early**



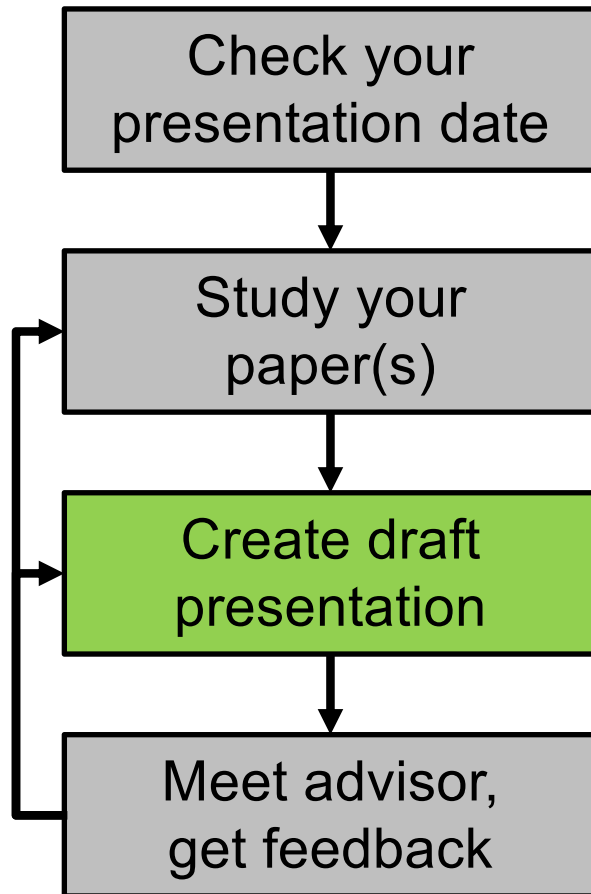
- Preparing a good presentation takes time
- So, please start early!

Preparing a Talk: **Study Paper**



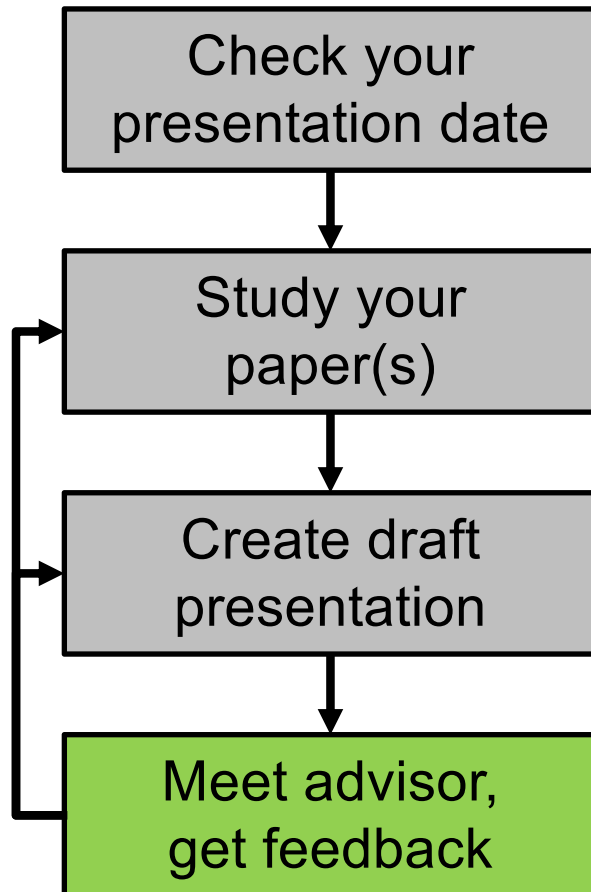
- 3 C's of reading
 - **Carefully**: look up terms, possibly read cited papers
 - **Critically**: find limitations, flaws
 - **Creatively**: think of improvements
- **Try examples** by hand
- **Try tools** if available
- Consult advisors for questions

Preparing a Talk: **Create Draft**



- Explain the work's **motivation**
- Clearly present the **technical solution** and **results**
 - Use **own examples**, not the ones in the paper
 - **Include a demo** if appropriate
- Discuss **limitations** or **improvements**
- Focus on **key concepts**
 - Do not present all the details

Preparing a Talk: **Get Feedback**



- **Prepare** for the meeting
 - Schedule early
 - Send slides in advance
 - Write down questions
- Do **address feedback**
 - Take notes
- **Meeting is mandatory!**
 - At least 1 week before the talk

Grading

- Quality of your presentation
 - How well did you understand the material?
 - How well did you present it?
 - How well did you answer the questions?
- Participation
 - Did you ask good questions?
 - Did you attend all sessions?
- We will take into account
 - the difficulty of the paper
 - suggestions you received from your TA advisor
 - time you had to prepare

Feedback

- Discuss your talk's strengths/weaknesses in-class
 - Let us know upfront if you prefer us not to
- Arrange meeting with your advisor to get feedback

Schedule

- Meet once per week with ~2 presentations each time
 - Next meeting on March 10th
- Detailed schedule will be published online shortly
 - <https://people.inf.ethz.ch/suz/teaching/263-2100-s20.html>
 - Including names of advisors/mentors

Your Talk: **Timing**

- 30 min for talk
 - 1.5 ~ 2 min per slide
- 15 min for Q&A and discussion
- The pace of talk is important
 - Too fast, the audience cannot follow
 - Too slow, people can get bored
- Practice your talk
 - Checkpoint after ~10 minutes



Your Talk: **Examples**

- Examples are crucial for understanding
 - Both yours and the audience's
 - Prepare your own examples
- Try to find a running example
 - For motivation, problem, solution
 - Explain in detail (takes time)
- Reduce code example to the absolute necessary
 - Most people hate reading code
 - Use visualizations

llvm bug autopsy

```

struct tiny { char c; char d; char e; };

void foo(struct tiny x) {
    if (x.c != 1) abort();
    if (x.e != 1) abort();
}

int main() {
    struct tiny s;
    s.c = 1; s.d = 1; s.e = 1;
    foo(s);
    return 0;
}
  
```

GVN: load struct using 32-bit load

SROA: read past the struct's end

remove

undefined behavior

```

$ clang -m32 -O0 test.c ; ./a.out
$ clang -m32 -O1 test.c ; ./a.out
Aborted (core dumped)
  
```

Your Talk: Design

descriptive
title

llvm bug autopsy

don't
overload
slide

```
struct tiny { char c; char d; char e; };
```

```
void foo(struct tiny x) {  
    if (x.c != 1) abort();  
    if (x.e != 1) abort();  
}
```

GVN: load struct
using 32-bit load

```
int main() {  
    struct tiny s;  
    s.c = 1; s.d = 1; s.e = 1;  
    foo(s);  
    return 0;  
}
```

SROA: read past
the struct's end

→
undefined
behavior

remove

visuals

large font
(> 20pt)

```
$ clang -m32 -O0 test.c ; ./a.out  
$ clang -m32 -O1 test.c ; ./a.out  
Aborted (core dumped)
```

Powerpoint vs. Latex

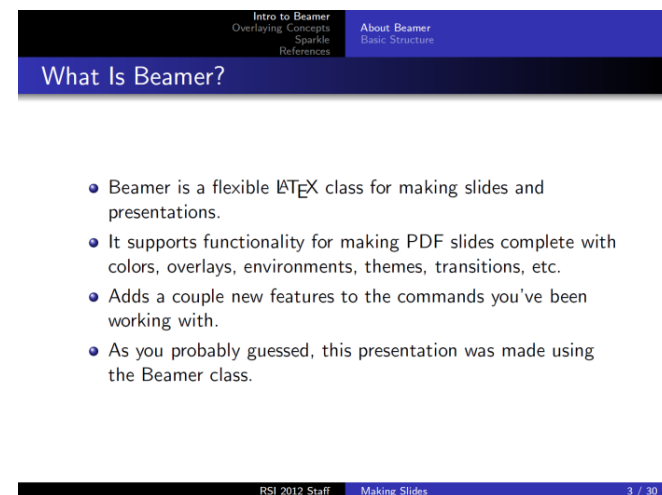
■ Powerpoint

- Visualizations and animations are easy
- Don't over-do it!



■ Latex

- Visualizations and animations are painful
- Don't under-do it!



Your Talk: **Avoid Frequent Mistakes**

- Don't try to present all details
 - Focus on the key messages:
motivation, problem, main idea, main result
- Don't stare at the screen or your laptop
 - Look at the audience
- Come prepared
 - Study the paper in depth
 - Rehearse your talk

References

Markus Püschel's small guide on giving talks

<http://www.inf.ethz.ch/personal/markusp/teaching/guides/guide-presentations.pdf>

Highly recommended!

Paper Pool for Spring 2020

1. [Coq Coq Correct! Verification of Type Checking and Erasure for Coq, in Coq](#). POPL 2020.
2. [Executable formal semantics for the POSIX shell](#). POPL 2020.
3. [Incorrectness Logic](#). POPL 2020.
4. [Debugging Inputs](#). ICSE 2020.
5. [Quickly Generating Diverse Valid Test Inputs with Reinforcement Learning](#). ICSE 2020.
6. [Adversarial sample detection for deep neural network through model mutation testing](#). ICSE 2019.
7. [Fuzz Testing based Data Augmentation to Improve Robustness of Deep Neural Networks](#). ICSE 2020.
8. [Testing DNN Image Classifier for Confusion & Bias Errors](#). ICSE 2020.
9. [TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing](#). ICML 2019.
10. [Defending Against Physically Realizable Attacks on Image Classification](#). ICLR 2020.
11. [Chopped Symbolic Execution](#). ICSE 2018.
12. [Parser-Directed Fuzzing](#). PLDI 2019.
13. [Initialize Once, Start Fast: Application Initialization at Build Time](#). OOPSLA 2019.
14. [Excelint: Automatically Finding Spreadsheet Formula Errors](#). OOPSLA 2018.
15. [Memento mori: dynamic allocation-site-based optimizations](#). ISMM 2015.
16. [Defining the Undefinedness of C](#). PLDI 2015.
17. [Efficient Lock-Free Durable Sets](#). OOPSLA 2019.
18. [Mesh: Compacting Memory Management for C/C++ Applications](#). PLDI 2019.
19. [Virtual Machine Design for Parallel Dynamic Programming Languages](#). OOPSLA 2018.
20. [REPT: Reverse Debugging of Failures in Deployed Software](#). OSDI 2018.
21. [Automated clustering and program repair for introductory programming assignments](#). PLDI 2018.
22. [A Systematic Literature Review of Automated Feedback Generation for Programming Exercises](#). TOCE 2019.
23. [Rethinking Debugging as Productive Failure for CS Education](#). SIGCSE 2019.
24. [Does syntax highlighting help programming novices?](#). ESE 23 (2018).
25. [What distinguishes great software engineers?](#). ESE 25 (2020).
26. [Finding Crash-Consistency Bugs with Bounded Black-Box Crash Testing](#). OSDI 2018.
27. [HyDiff: Hybrid Differential Software Analysis](#). ICSE 2020.
28. [Causal Testing: Understanding Defects' Root Causes](#). ICSE 2020.
29. [Synthesizing Database Programs for Schema Refactoring](#). PLDI 2019.
30. [Coverage Guided, Property Based Testing](#). OOPSLA 2019.
31. [SLING: Using Dynamic Analysis to Infer Program Invariants in Separation Logic](#). PLDI 2019.
32. [HOPPITY: LEARNING GRAPH TRANSFORMATIONS TO DETECT AND FIX BUGS IN PROGRAMS](#). ICLR 2020.
33. [Scalable Taint Specification Inference with Big Code](#). PLDI 2019.
34. [code2vec: Learning Distributed Representations of Code](#). POPL 2019.
35. [Code vectors: understanding programs through embedded abstracted symbolic traces](#). ESEC/FSE 2018.