Introduction & Overview

CS350 Introduction to Software Engineering

Shin Yoo

CS350 Introduction to Software Engineering

Spring 2023

- Lecturer: Shin Yoo (first timer for CS350, be gentle)
    - Office: E3-1 #2405
    - shin.yoo@kaist.ac.kr
- Course Webpage: https://coinse.kaist.ac.kr/teaching/2023/cs350/
- TAs
    - Gabin An (agb94@kaist.ac.kr), Somin Kim (thaxls@kaist.ac.kr), Hyunseok Lee (christmas@kaist.ac.kr)

CS350 Introduction to Software Engineering
Spring 2023

- Grading
  - Assignments: 40%
  - Project: 40%
  - Quiz: 40%
- There is no exam, and there will be no lectures during the exam weeks.
- Requirements
  - Reasonable programming skills

Project

CS350 | Spring 2023

- Team-based: form a 4-member team
- You will be both developers and clients :)
  - As the client, you will propose a project (by writing requirement specs)
  - Teams will sign up for projects
  - As the developer, you will propose how you will make it (by writing design document)
  - You will sync up multiple times and report your interactions :)

Assignments

CS350 | Spring 2023

- Assignment 1: Essay on "The Mythical Man Month" - we will read a classic in SE literature and consider whether it still holds good.
- Assignment 2: "Find a Bug Day" - we will find and document real world SW bugs that affected us, and imagine how they could have been prevented.
- Assignment 3: "Git Challenge" - we will improve our git skills and solve a small quiz.
- Assignment 4: "Future of Software Engineering" - we will try to decide whether software engineers will go extinct thanks to AI.
- See the course webpage for more details including due dates.

Communications

CS350 | Spring 2023

- We will use KLMS only for assignment submission.
- Lecture materials will be published on the course webpage.
- Class communication will take place on Slack
  - You are required to join cs350spring2023kaist.slack.com
  - Set your profile name as “your full English name (your student number)”
  - Use channels based on their names; please do not hesitate to ask questions

DOs and DON’Ts

CS350 | Spring 2023

- Please engage during the class - speak up, ask questions, etc.
    - There is no “grading” for participation - no pressure to say clever things :p
    - I mean…. why not?
- DO NOT CHEAT - you will **fail** the course, no exception. By cheating I refer to:
    - Copying other people’s text/code (or simply generating text/code using AI, unless you are asked to do by me)
    - Sharing your solution publicly without any safeguard

Any questions?

What is software engineering?

How is it different from programming?

Software Engineering?

- “The application of a systematic, disciplined, quantifiable approach to development, operation and maintenance of software” (IEEE Systems and Software Engineering Vocabulary, 2010)
- “a systematic engineering approach to software development” (Wikipedia, https://en.wikipedia.org/wiki/Software_engineering)

Software Crisis

- As long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.
— Edsger Dijkstra, The Humble Programmer, Communications of the ACM, 1972

NATO Conference 1968

(Dijkstra was also there)

- *The phrase 'software engineering' was <u>deliberately chosen as being provocative</u>, in implying the need for software manufacture to be based on the types of theoretical foundations and practical disciplines that are traditional in the established branches of engineering.*
  - Editors of the proceedings of Software Engineering (1968)

NATO Conference 1968

(Dijkstra was also there)

- The conference proceedings is a fascinating read: you can actually read the whole thing in a nicely typeset PDF: https://www.scrummanager.com/files/nato1968e.pdf

- There is also a very helpful blog post by Logan Mortimer, a software engineer in Melbourne: https://isthisit.nz/posts/2022/1968-nato-software-engineering-conference/

- Many interesting ideas and questions. Are software to be “manufactured” like other engineering products? How do we measure progress? How do we scale up to hundreds and thousands of people building the same system together?

How is SW like other engineering product?

How is it different from other engineering products?

Essential Properties of SW

as pointed out by Brooks Jr. in 1987

- Complexity
  - More complex than any other human constructs for their size (no two parts are identical, because we would factor them out)
  - Scaling up does not mean making the same thing larger
- Conformity
  - Physicists firmly believe that there is a unified theory of things
  - Most of complexity in SW is arbitrary - SW has to conform to countless many things (institutions, systems, users, regulations…)

Essential Properties of SW

as pointed out by Brooks Jr. in 1987

- Changeability
  - Compared to other engineering products, SW is more frequently pressured to change
  - If a software system is useful, people will try new edge cases at the borderline of the original domain
  - A successful software can outlive the underlying hardware, and therefore has to adapt
- Invisibility
  - SW cannot be easily embedded in space and therefore has no useful geometric representation
  - Without visual representation, communication and design becomes much more difficult

“No Silver Bullet”

- “Of all the monsters that fill the nightmares of our folklore, none terrify more than werewolves, because they transform unexpectedly from the familiar into horrors. (…) (software) is usually innocent and straightforward, but is capable of becoming a monster of missed schedules, blown budgets, and flawed products.” - Brooks Jr.

“No Silver Bullet”

- Brooks Jr. argues that there is no silver bullet that will kill the SW werewolf: the difficulties are inherent in SW, and no single technique will solve all of these.
- In the same titled essay, Brooks Jr. examines some candidates for the silver bullet (back in 1987)

Hopes for the silver
(or the lack thereof)

- “Ada and other high level languages”: 🤠
- “Object-oriented programming”: OO paradigm has survived, but probably not THE only existing paradigm. The advanced type system he mentions is not mainstream yet…
- “AI and Expert System”: a very complicated matter… but is it the silver bullet?
- “Automatic Programming”: that is, synthesising program from the specifications… Large Language Models are in some way doing this, so part of the above
- “Graphical Programming”: okay, my 10 year old kid is graduating from Scratch to Python soon… :)

Hopes for the silver

(or the lack thereof)

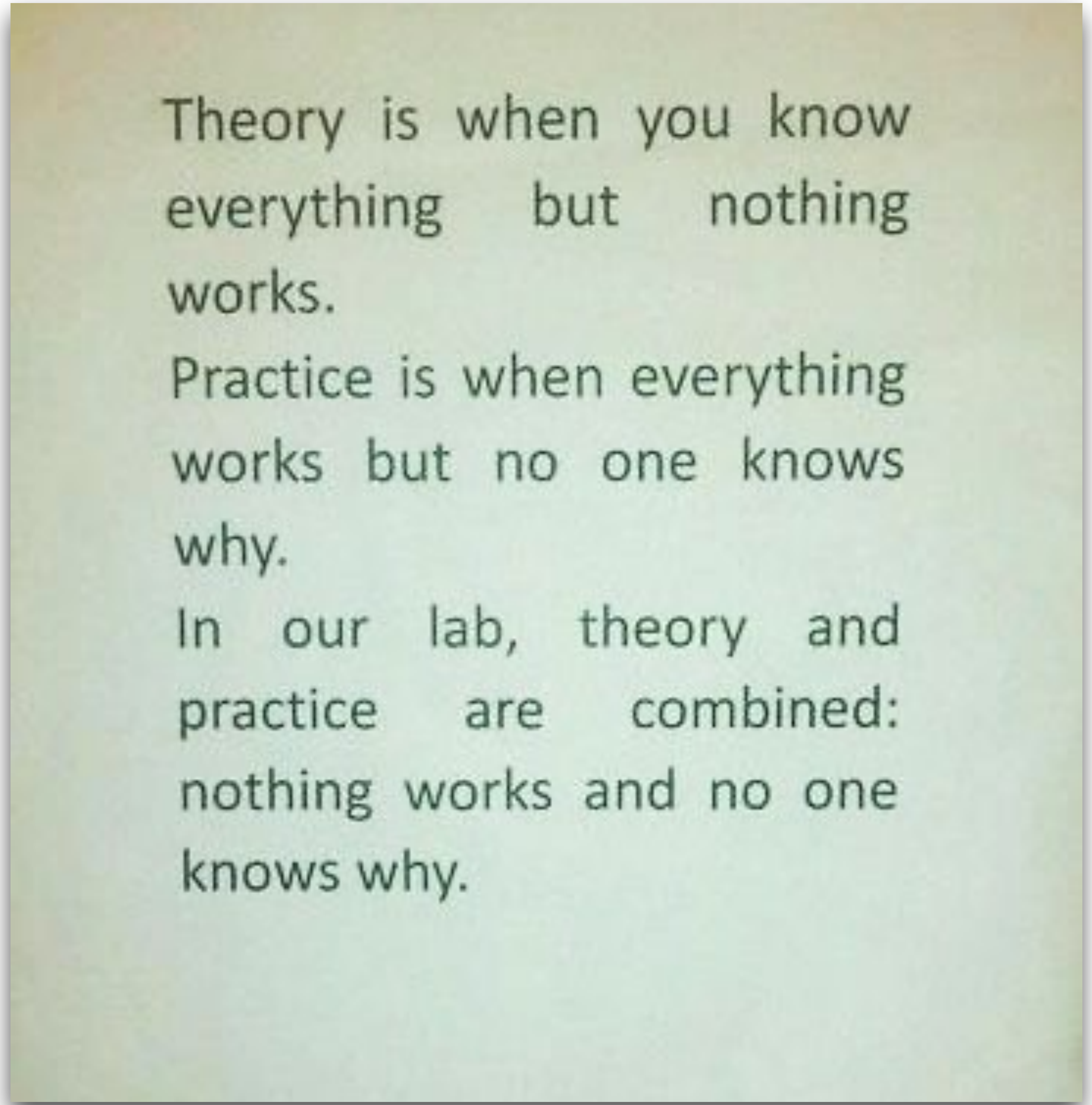
- “Graphical Programming”: okay, my 10 year old kid is graduating from Scratch to Python soon… :)
- “Program Verification”: i.e., the art of “proving” that the program does not have a bug - he says it would not really scale (which is true to this day)
- “Environment and tools”: we have IDEs and build systems that are beyond the imagination of 1987 yet the SW problem has not gone away
- “Workstations”: “Well, how many MIPS can one use fruitfully? Compiling could stand a boost, but a factor of 10 in machine speed would surely leave thinking time the dominant activity in the programmer’s day” :)

“If SW is inherently and arbitrarily complex, does not conform to laws of nature or any single theory, and if none of the technical advances during the last 30+ years solved it, why should we study software engineering?”

🥺

About SE and Theory 🥺

- It may not be entirely untrue that there is no single general unifying theory of how to build a perfect SW system… apart from the mantra of "think very hard and do it very well"
- Are we doomed?



Theory is when you know everything but nothing works.
Practice is when everything works but no one knows why.
In our lab, theory and practice are combined: nothing works and no one knows why.

About SE and Theory

or “How I learned to stop worrying and love SE 🙃”

- Given the complexity and diversity of SW systems, it is almost trivially guaranteed that no single theory can solve everything.
- Think of SE as a multiverse of theories and underlying sub-fields - in fact, it is such an umbrella term that I myself do not know where SE starts and ends.
- Each theory guides some best practice - we need to learn which works where.

De-mystifying CS350

- *"It is a humanities course - nothing is technical"*: believe me, most of the topics we will go through has much more technical difficulties hidden underneath, we just cannot cover all of them in an introductory course!

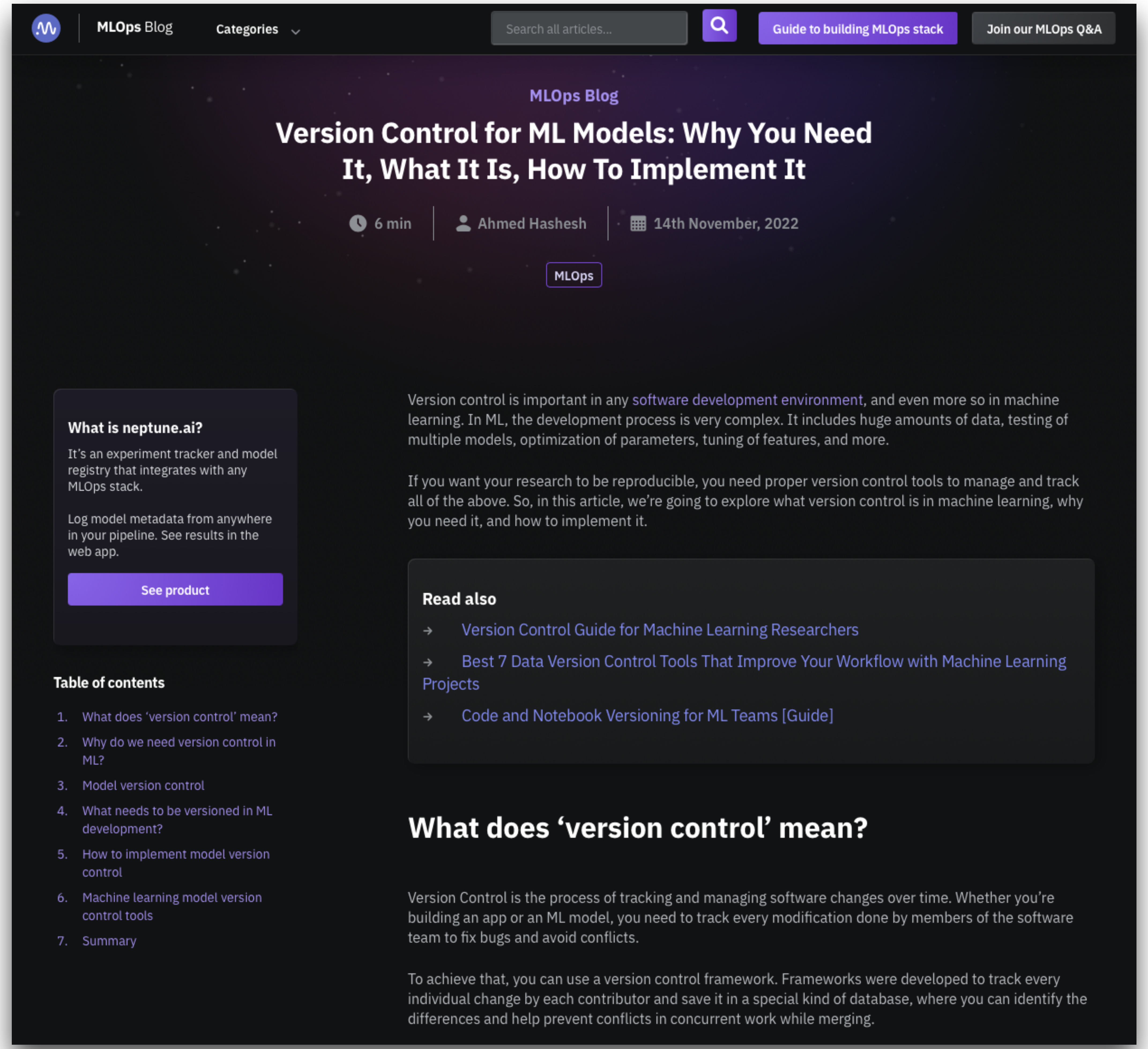
- Requirement Engineering (Formal Specification), Testing and Analysis (Verification, Program Analysis, Testing Automation), Maintenance (Predictive Modelling, Data Mining for SW Development)…

De-mystifying CS350

- *"It is boring - you only ever write documentations"*: there ARE still systems for which you really need to do all of those. It is just that, recently, we have so much more software that are casual to use AND develop!

As long as you make software, SE is relevant

(regardless of the domain: ML, CV, NLP, etc…



MLOps Blog

Version Control for ML Models: Why You Need It, What It Is, How To Implement It

6 min | Ahmed Hashesh | 14th November, 2022

MLOps

What is neptune.ai?

It's an experiment tracker and model registry that integrates with any MLOps stack.

Log model metadata from anywhere in your pipeline. See results in the web app.

See product

Table of contents

Version control is important in any software development environment, and even more so in machine learning. In ML, the development process is very complex. It includes huge amounts of data, testing of multiple models, optimization of parameters, tuning of features, and more.

If you want your research to be reproducible, you need proper version control tools to manage and track all of the above. So, in this article, we're going to explore what version control is in machine learning, why you need it, and how to implement it.

Read also

→ Version Control Guide for Machine Learning Researchers

→ Best 7 Data Version Control Tools That Improve Your Workflow with Machine Learning Projects

→ Code and Notebook Versioning for ML Teams [Guide]

What does 'version control' mean?

Version Control is the process of tracking and managing software changes over time. Whether you're building an app or an ML model, you need to track every modification done by members of the software team to fix bugs and avoid conflicts.

To achieve that, you can use a version control framework. Frameworks were developed to track every individual change by each contributor and save it in a special kind of database, where you can identify the differences and help prevent conflicts in concurrent work while merging.

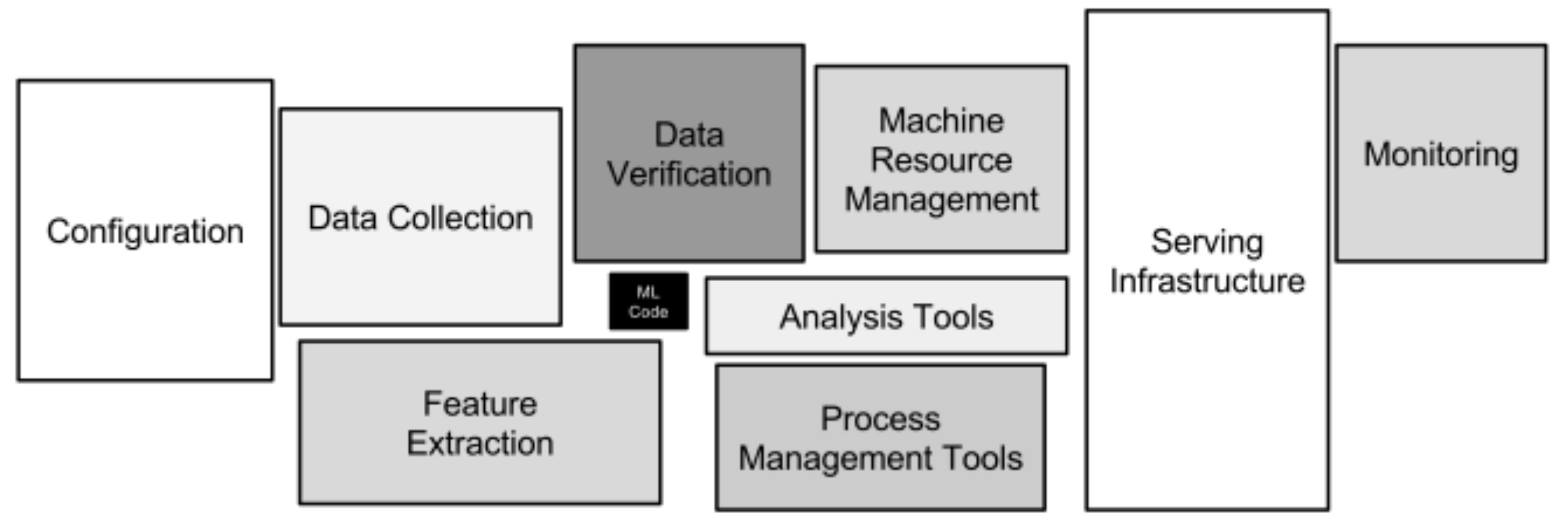
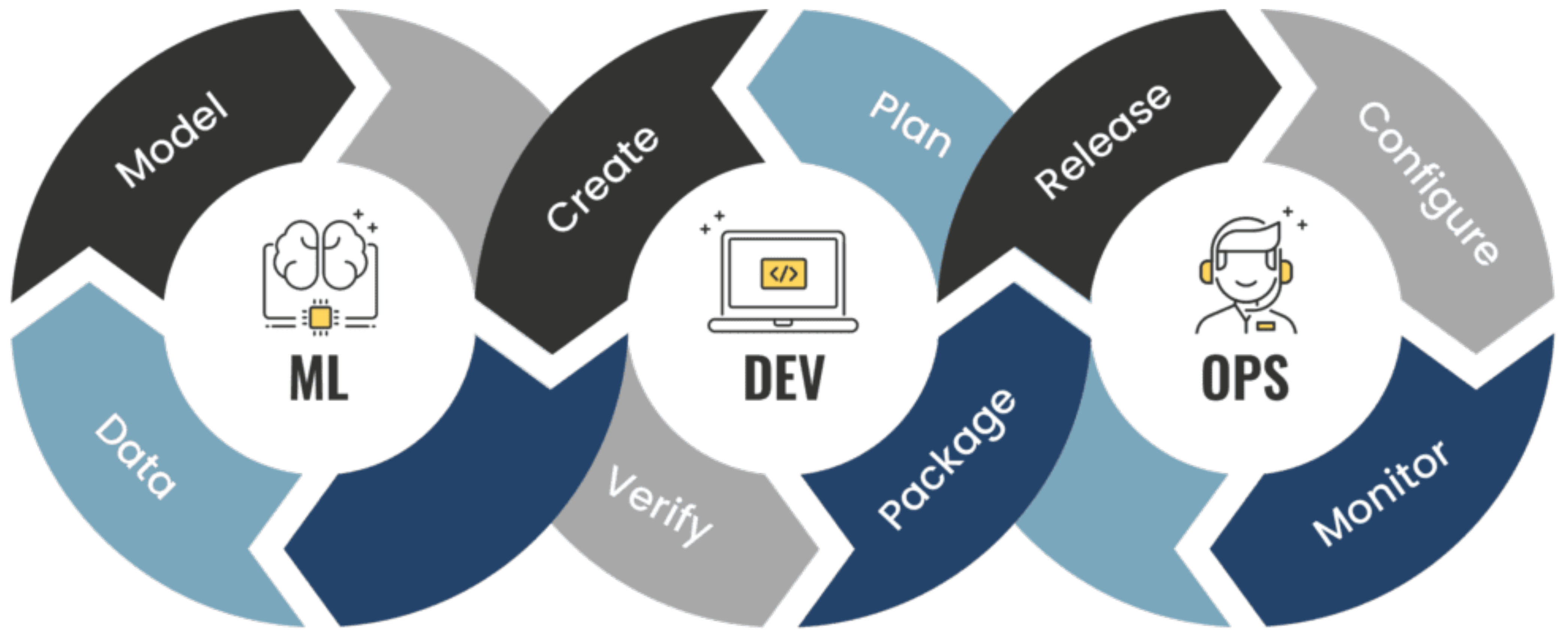
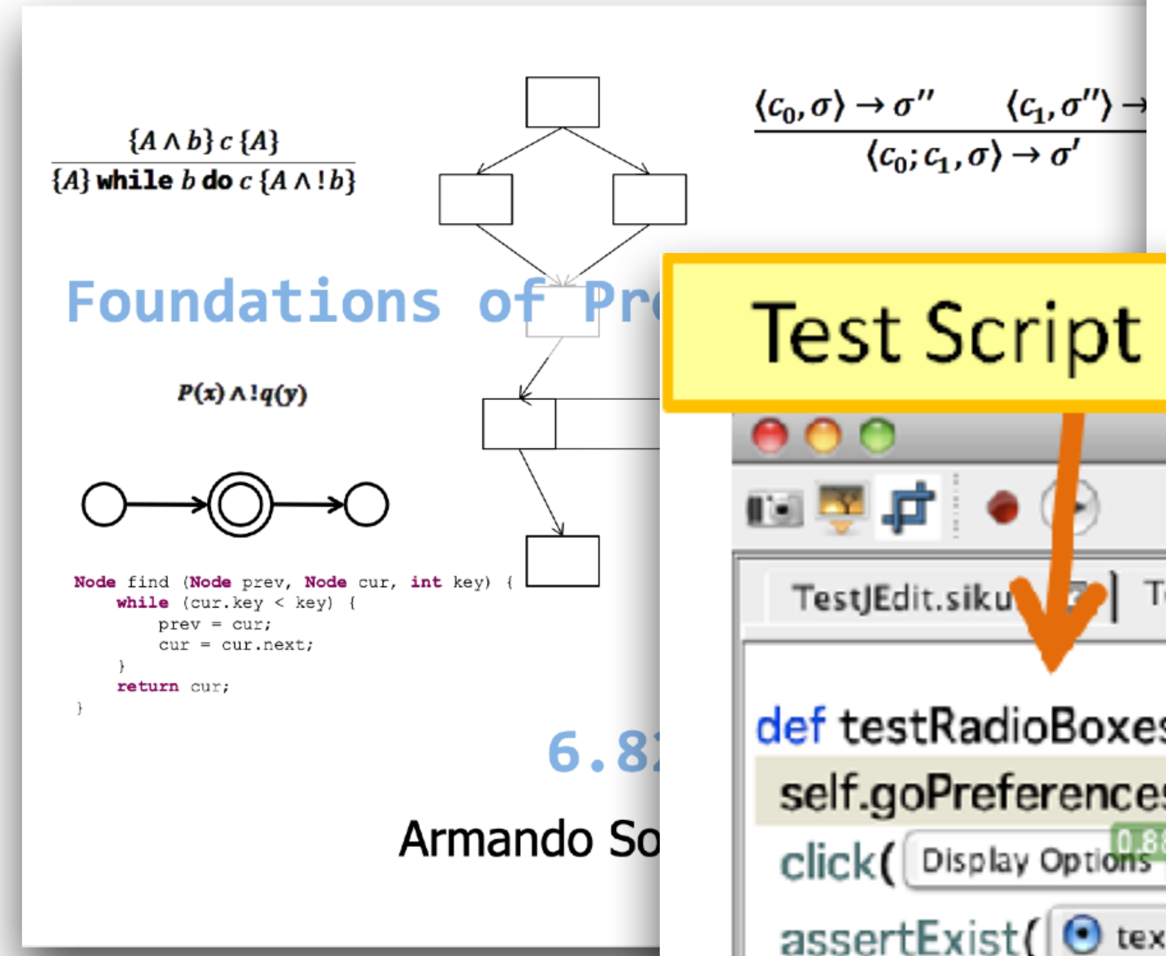


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.



All other CS principles contribute to SE

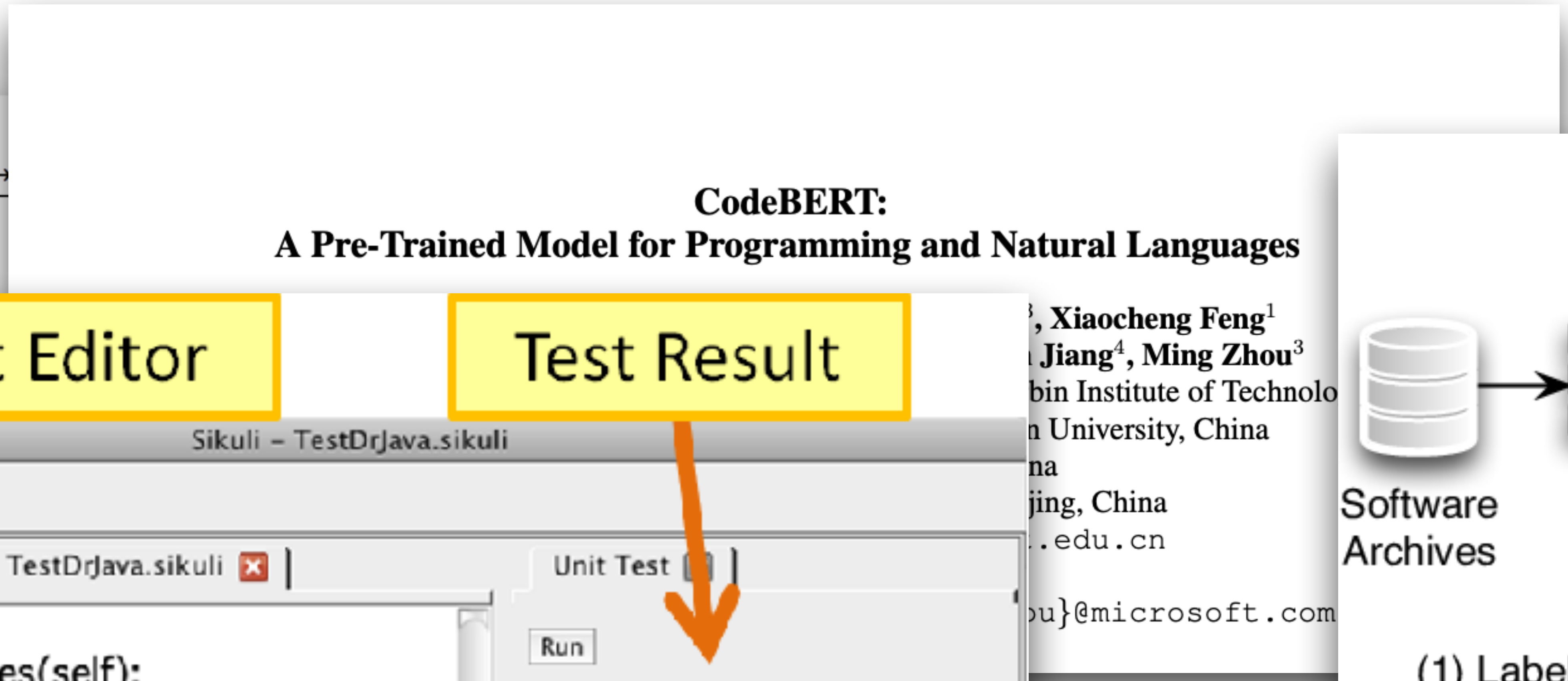
(we welcome anything that works)



$\{A \wedge b\}\, c\, \{A\}$ / $\{A\}$ while b do c $\{A \wedge !b\}$
$\langle c_0, \sigma\rangle \to \sigma''$ $\langle c_1, \sigma''\rangle \to$ / $\langle c_0; c_1, \sigma\rangle \to \sigma'$
Foundations of Pr
$P(x) \wedge !q(y)$
6.8
Armando So

```
Node find (Node prev, Node cur, int key) {
  while (cur.key < key) {
    prev = cur;
    cur = cur.next;
  }
  return cur;
}
```

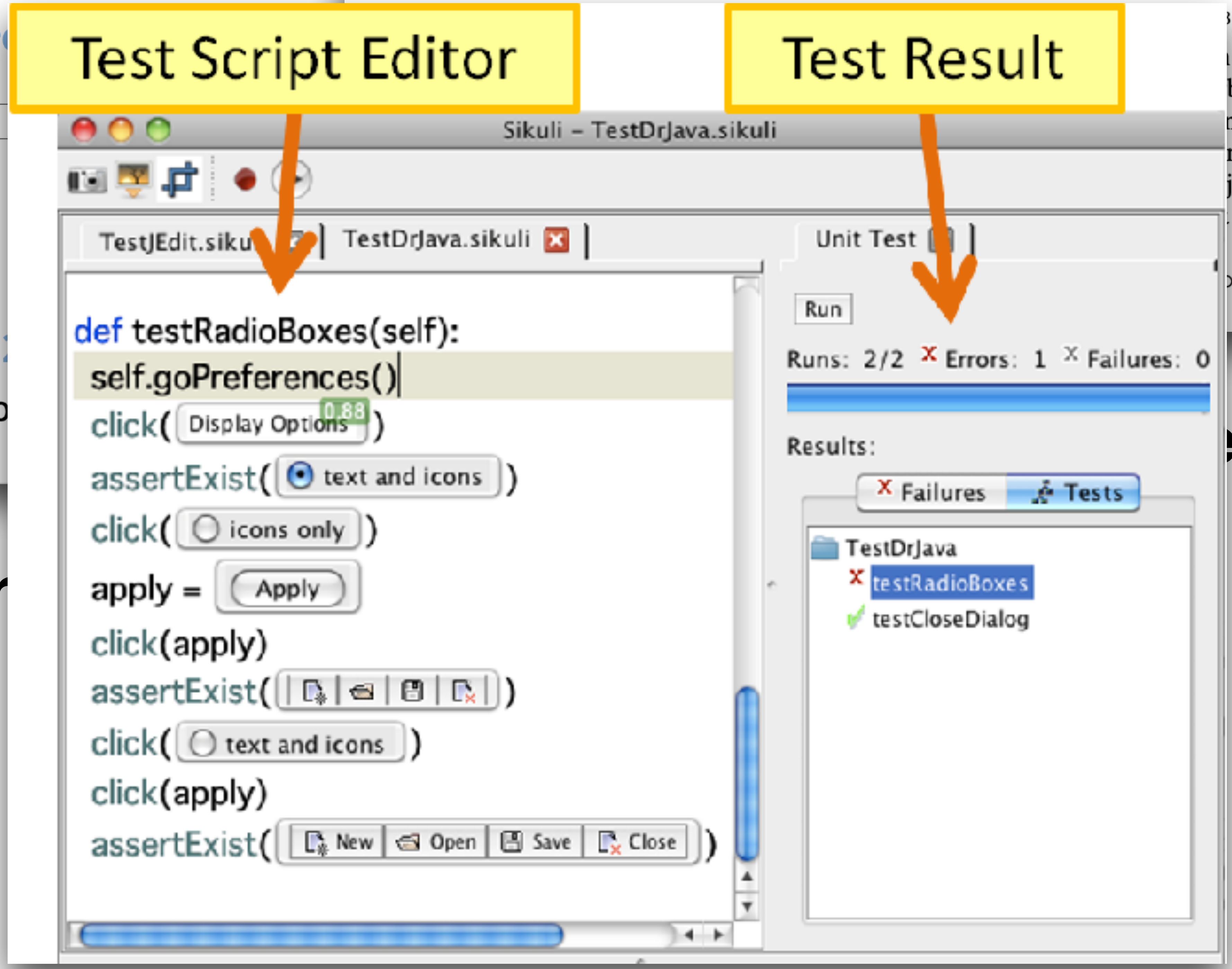
PL —> Progr



CodeBERT:
A Pre-Trained Model for Programming and Natural Languages

, Xiaocheng Feng[1]
Jiang[4], Ming Zhou[3]
bin Institute of Technolo
n University, China
na
jing, China
.edu.cn
u}@microsoft.com

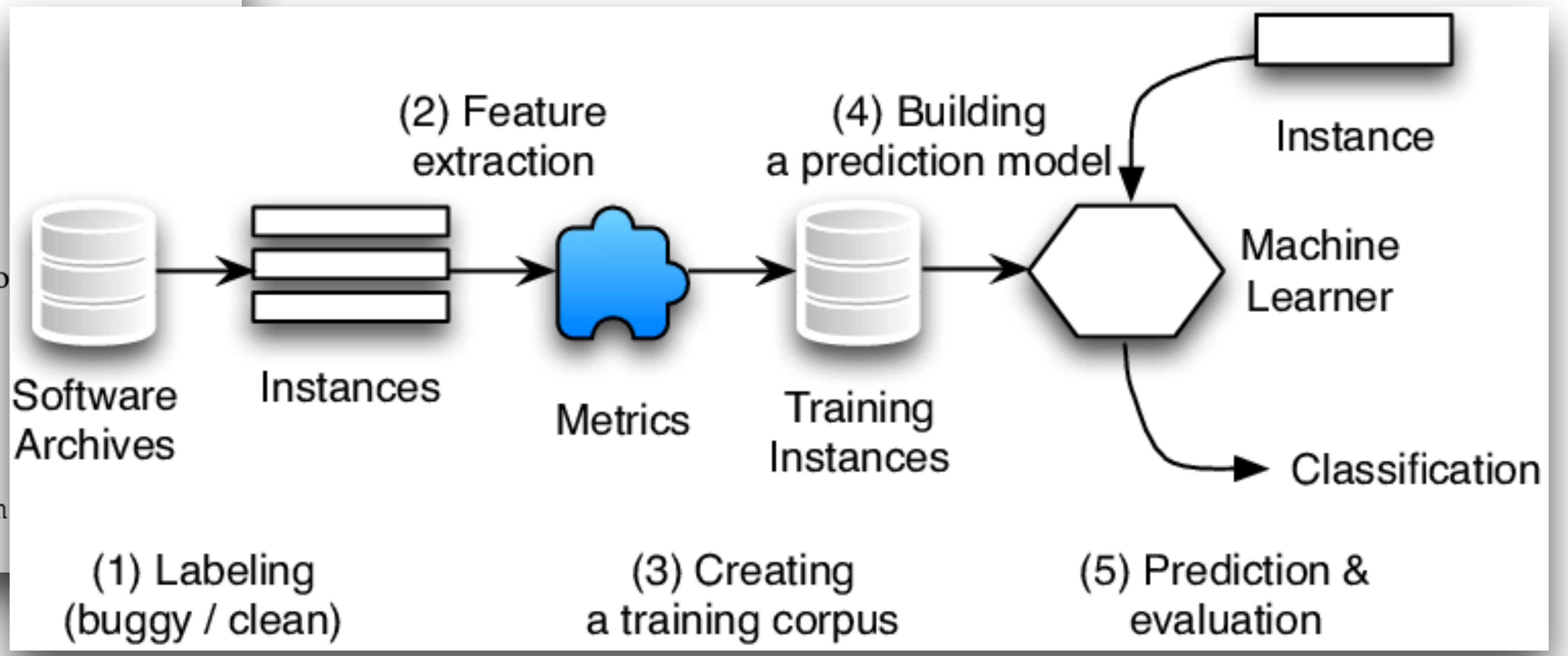
eration



```
def testRadioBoxes(self):
  self.goPreferences()
  click( Display Options 0.88 )
  assertExist( text and icons )
  click( icons only )
  apply = Apply
  click(apply)
  assertExist( ... )
  click( text and icons )
  click(apply)
  assertExist( New Open Save Close )
```

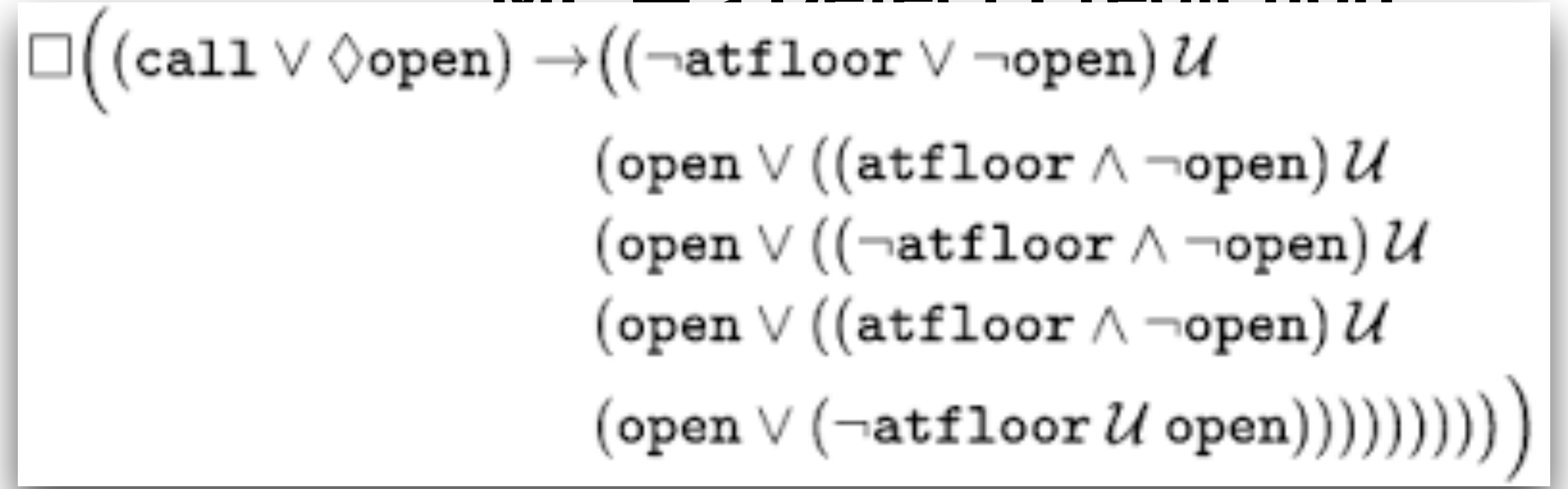
```
Runs: 2/2  Errors: 1  Failures: 0
Results:
TestDrJava
  testRadioBoxes
  testCloseDialog
```

Vision —> GUI Testing



Software Archives
Instances
Metrics
Training Instances
Machine Learner
Instance
Classification
(1) Labeling (buggy / clean)
(2) Feature extraction
(3) Creating a training corpus
(4) Building a prediction model
(5) Prediction & evaluation

ML —> Defect Prediction



$$\Box\Big((\texttt{call} \vee \Diamond\texttt{open}) \rightarrow ((\neg\texttt{atfloor} \vee \neg\texttt{open})\,\mathcal{U}\,(\texttt{open} \vee ((\texttt{atfloor} \wedge \neg\texttt{open})\,\mathcal{U}\,(\texttt{open} \vee ((\neg\texttt{atfloor} \wedge \neg\texttt{open})\,\mathcal{U}\,(\texttt{open} \vee ((\texttt{atfloor} \wedge \neg\texttt{open})\,\mathcal{U}\,(\texttt{open} \vee (\neg\texttt{atfloor}\,\mathcal{U}\,\texttt{open})))))))))\Big)$$

Logic Theory —> Model Checking

Summary

- SW is probably the most complex engineering artefact that humans build.
- Building SW well requires more than excellent programming skills (although it helps).
- We will spend the semester thinking about known problems, and get ourselves familiarised with widely accepted responses to those problems.
- It may not be easy bit I hope it will be fun :)