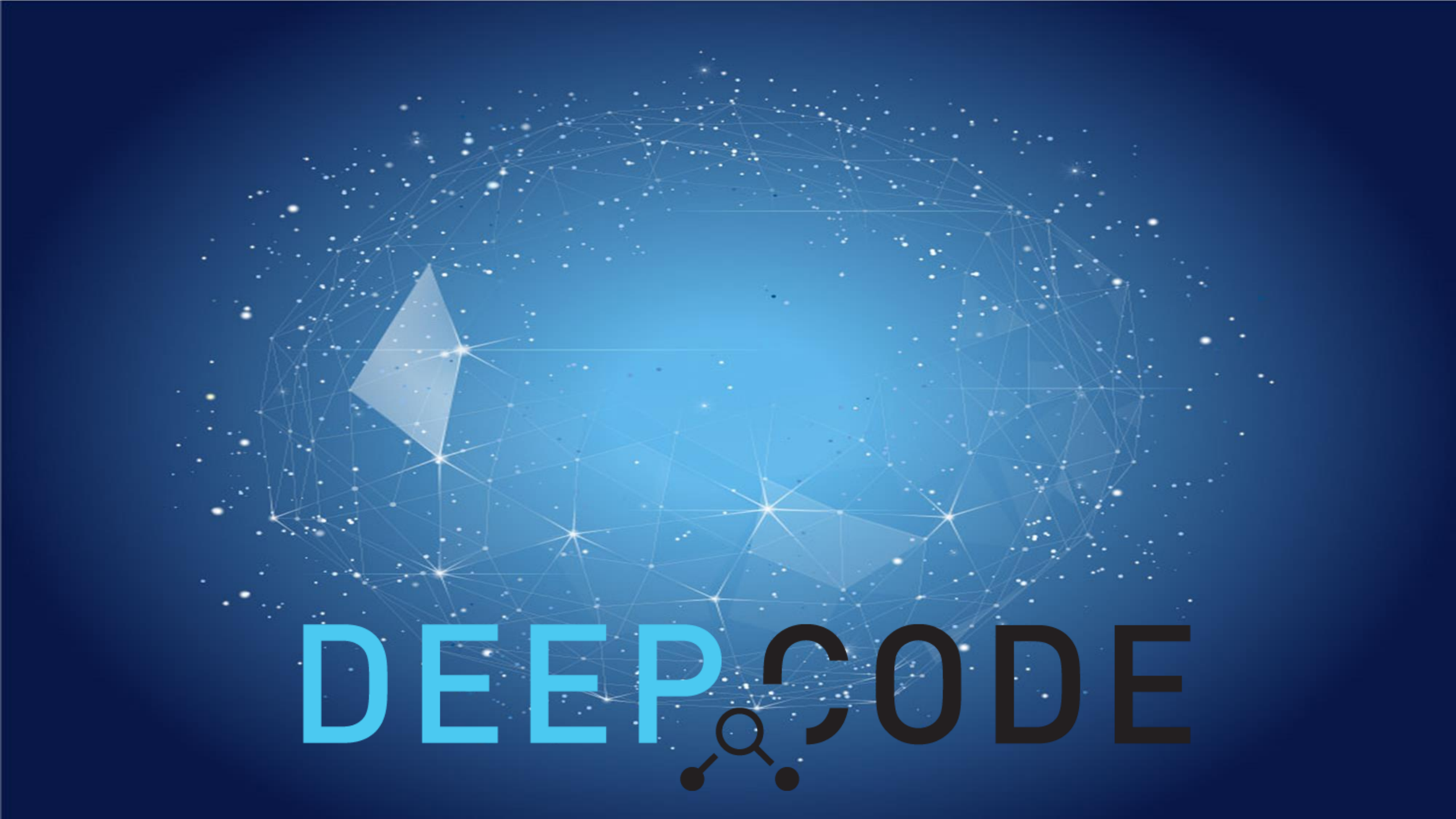

DEEPCODE

PRACTICAL STATIC PROGRAM ANALYSIS

Frank Fischer / @cu_0xff

JAVA USER GROUP CH

DEEPCODE

About me or us…

US

- Found 2016 / Spin off by ETH Zürich / DeepTech Startup
- People – see picture and in the room

ME

- For - like - ever in IT
- Worked for Microsoft, Google, Deutsche Telekom…
- MSc IT & MSc Physics

DEEPCODE

Agenda

- Intro, some stats, and motivation
- Some theory refresher (semantic vs syntax, ML vs Symbolic AI, representing code in graphs)
- Textual processing – the very basic tools and ML
- Linters and AST
- Event Graphs and Symbolic AI -> Demo
- Dos and Don'ts – how to build your tool stack
- Outlook – what to expect more
- Call to action

Motivator

DEEPCODE

Bloomberg

Politics

F-35's Gun That Can't Shoot Straight Adds to Its Roster of Flaws

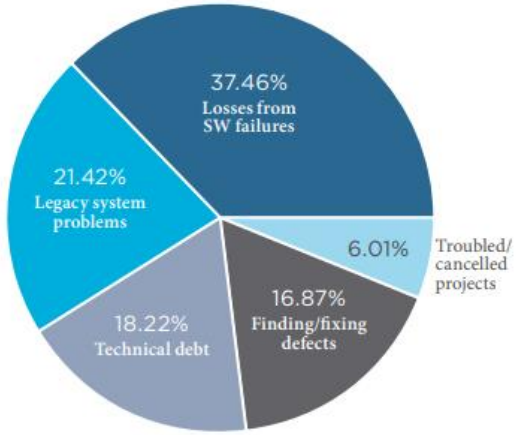
By Anthony Capaccio
January 30, 2020, 10:00 AM GMT+1

- Annual Pentagon testing report also finds 873 software issues
- It cites 13 'must-fix' items before $22 billion upgrade phase

Just to state the obvious…

CISQ Consortium for IT Software Quality | *The Cost of Poor Software Quality in the US: A 2018 Report*

In summary, the cost of poor quality software in the US in 2018 is approximately **$2.84 trillion**, the main components of which are seen in the following graph. If we remove the future cost of technical debt, the total becomes $2.26 trillion. For simplification, the various cost categories are, at this time, assumed to be mutually



GDP USA	US$ 19,390,000,000,000
GDP Germany	US$ 3,677,000,000,000
GDP Switzerland	US$ 678,900,000,000
Damage due to Poor SW Quality in the US/'18	**US$ 2,840,000,000,000**

DEEPCODE

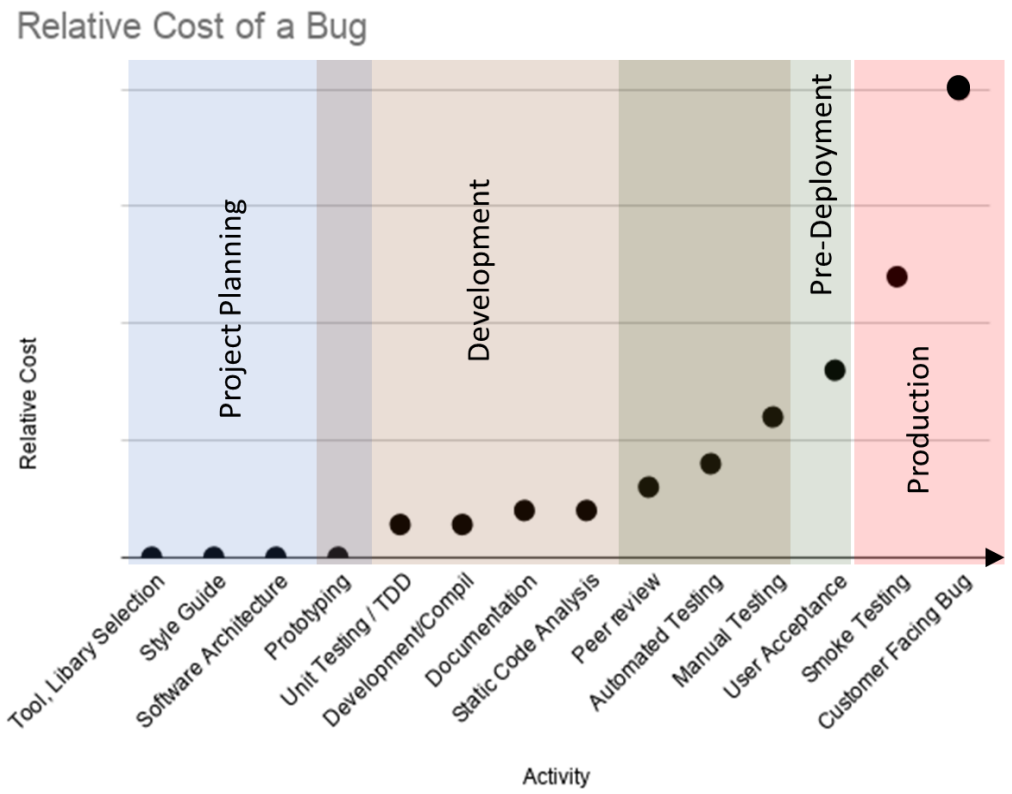
Software Quality – What is it?

The ISO/IEC Standard 25010:2011

Functional Suitability	Performance Efficiency	Compatibility	Usability	Reliability	Security	Maintainability	Portability
•Functional Completeness •Functional Correctness •Functional Appropriateness	•Time Behavior •Resource Use •Capacity	•Coexistance •Interoperability	•Appropriateness Recognizability •Learnability •User-error Protection •User-interface Aesthetics •Accessiblity	•Maturity •Availability •Fault Tolerance •Recoverability	•Confidentiality •Integrity •Nonrepudiation •Accountability •Authenticity	•Modularity •Reusability •Analyzability •Modifiability •Testability	•Adaptability •Installability •Replaceability

DEEPCODE

Code Quality Best Practices

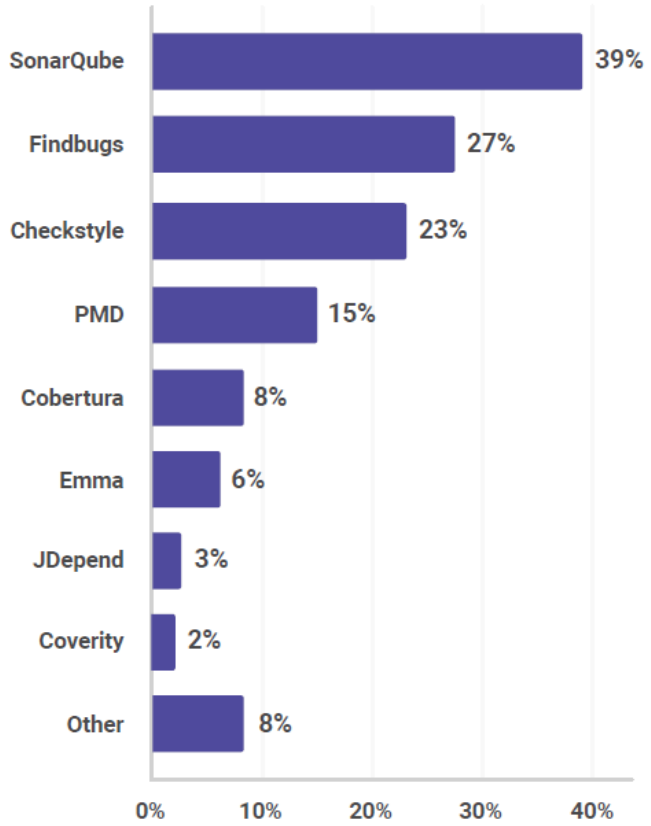


Static Program Analysis:
Analysing software without actually running it.

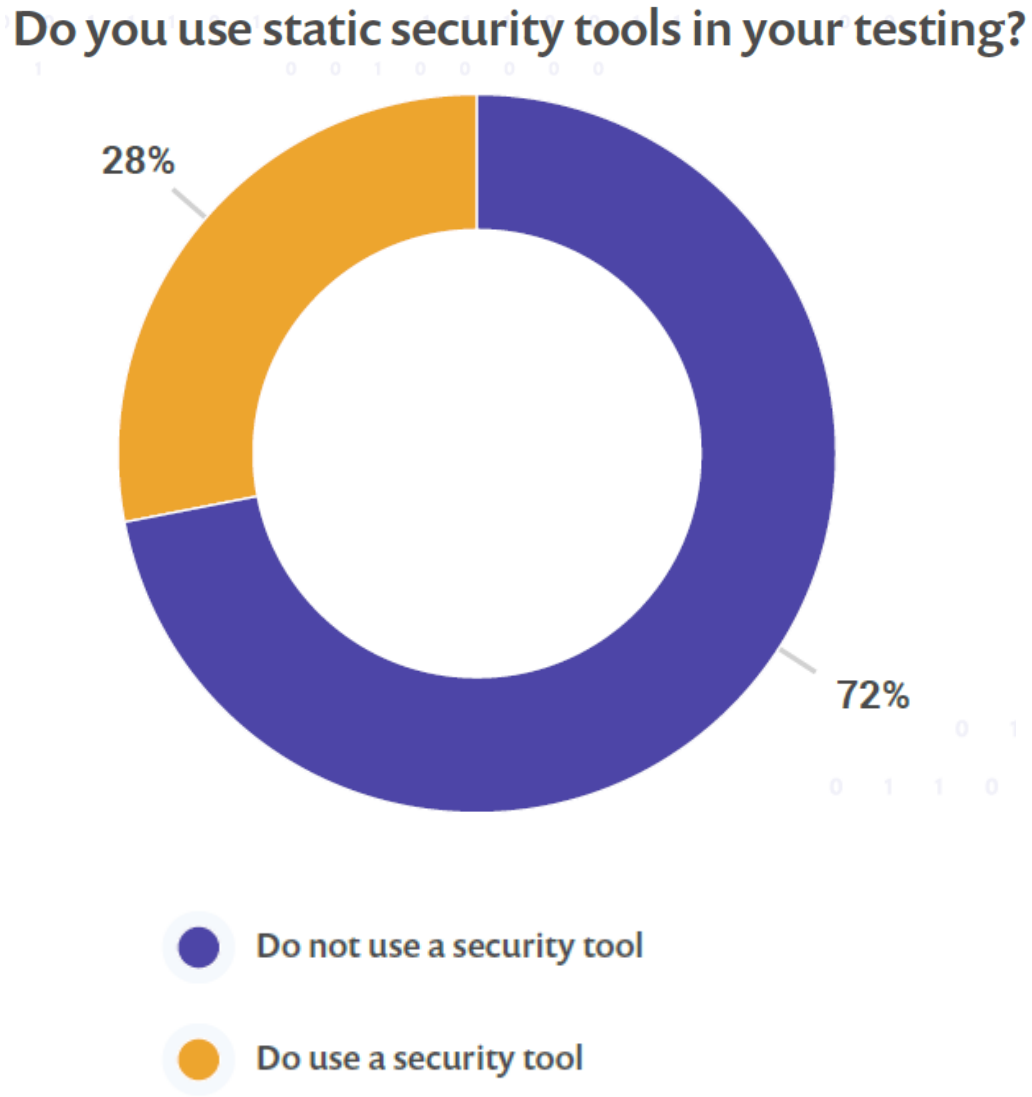
Revision Control	Git, svn, mercurial
Testing	Unit/integration/UI/ penetration tests
Code Review	Pull requests, Phabricator, Gerrit
Static Analysis	OUR FOCUS TODAY
Instrumentation / Monitoring	Java.lang.instrument, sqreen

DEEPCODE

Use of Static Program Analysis



36% said NONE



Source: Snyk "JVM Ecosystem Report – 2018"

DEEPCODE

Today's Limitations

Performance

Inter-procedural or inter-file is extremely resource intense

Quality

Recall versus Precision, types of bugs

Polyglot

Mix of languages, tools need to adapt

Linear Scaling of Rules

Handcrafting rules by experts scales linearly at best

Expressiveness

Found issues need to be explanatory and actionable

API Use

Ever changing libraries (if they would be annotated ☺)

DEEPCODE

Refreshers

Theory Refresher – Part 1: Semantic, Syntax

```
public class testmain{

    static Double p(Double x) { return x+x+x; }

    public static void main(String args[]) {
        if(p(0.1) == 0.3)
            System.out.println("Same");
        else
            System.out.println("Not the same");
    }
}
```

What is a valid program? Syntax and Semantic

Syntax: Correct in the sense of the **grammar**

Semantic: Reflects the **meaning**

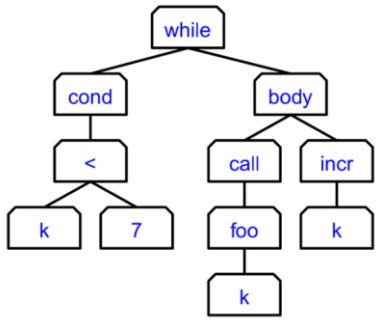
Typical Semantic Issues in Java:

Implicit Type Conversion, Precision, Operator Overloading

DEEPCODE

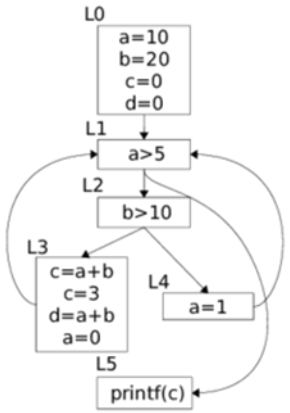
Theory Refresher – Part 2: Code in Graphs

```
while (k<7) {
   foo(k);
   k++;
}
```



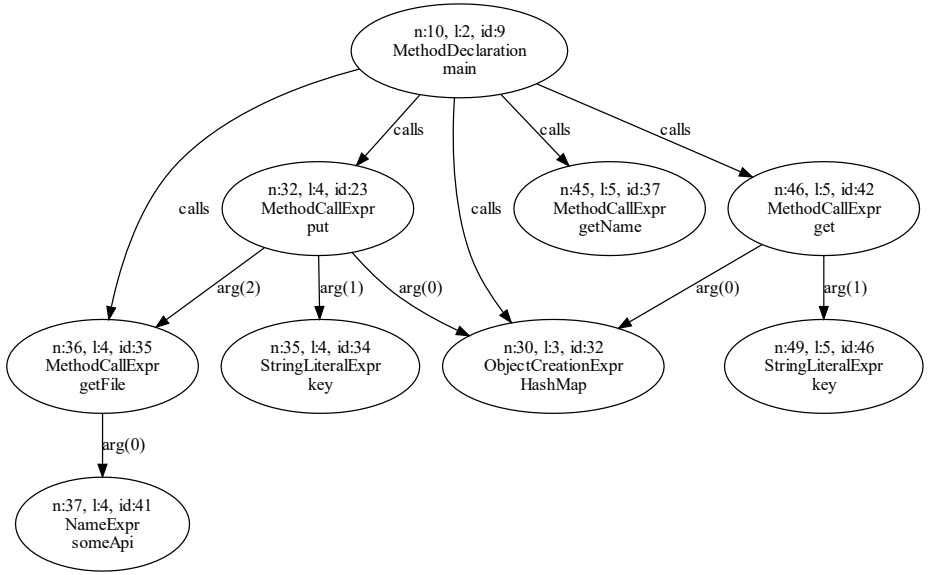
Abstract Syntax Tree

```
a=10;
b=20
c=0;
d=0;
L1:
if(a>5){
  if(b>10){
    c=a+b;
    c=3;
    d=a+b;
    a=0;
    goto L1;
  }
  else{
    a=1;
    goto L1;
  }
}
else{
  printf("%d\n",c);
}
```



Control Flow Graph

```
class X {
  public void main() {
    Map<String, File> map = new HashMap<>();
    map.put("key", someApi.getFile());
    String name = map.get("key").getName();
  }
}
```



Event Graph

Graphs serve as intermediate representations that expose different aspects (flow-sensitive, context-sensitive, path-sensitive)

DEEPCODE

Theory Refresher – Part 3: Symbolic vs Sub-symbolic AI

Symbolic

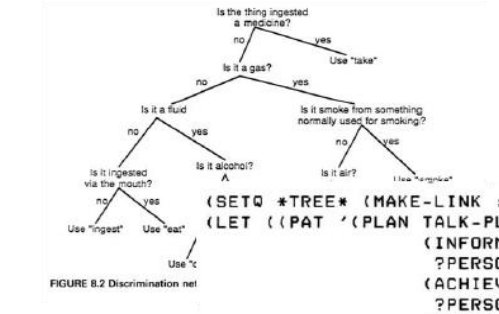
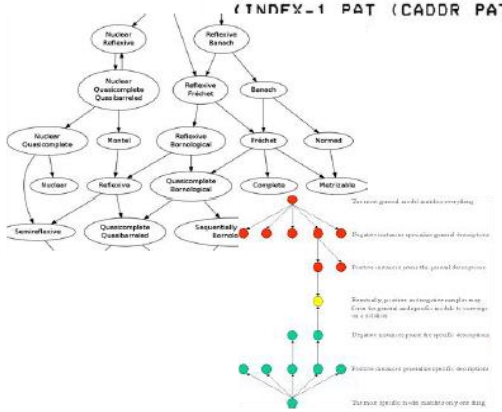
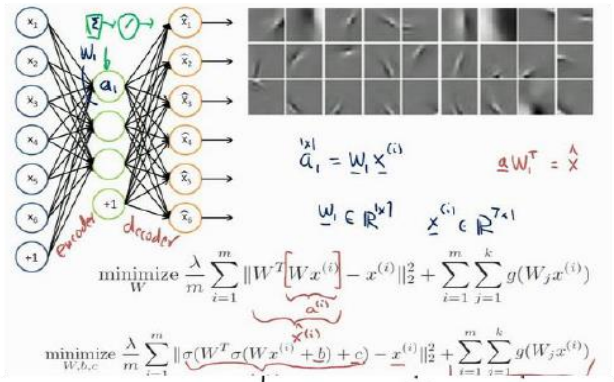


FIGURE 8.2 Discrimination net

```
(SETQ *TREE* (MAKE-LINK :KEY '*TOP*))
(LET ((PAT '(PLAN TALK-PLAN
                  (INFORM ?PERSON1
                   ?PERSON2 ?FACT)
                  (ACHIEVE (AT
                   ?PERSON1 ?PERSON2))
                  (SAY ?PERSON1 ?FACT))))
(INDEX-1 PAT (CADDR PAT) *TREE*))
```

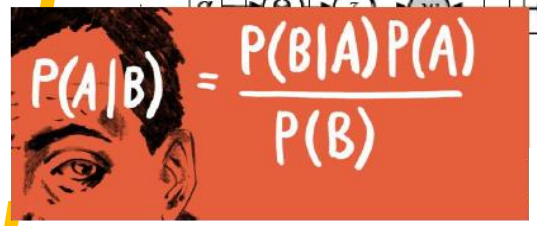


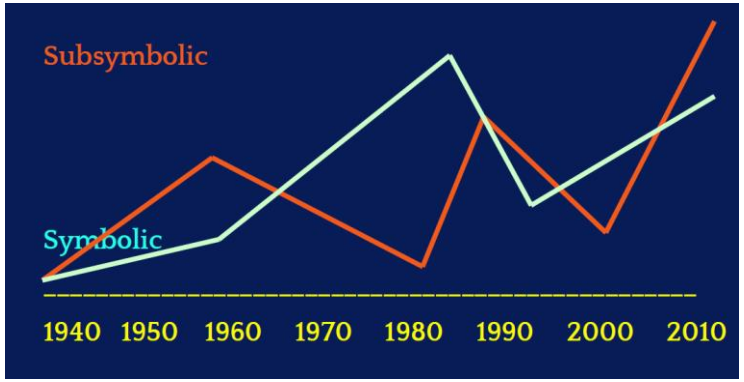
Sub-Symbolic



$$\underset{W}{\text{minimize}} \frac{\lambda}{m}\sum_{i=1}^{m}\|W^T[Wx^{(i)}] - x^{(i)}\|_2^2 + \sum_{i=1}^{m}\sum_{j=1}^{k} g(W_j x^{(i)})$$

$$\underset{W,b,c}{\text{minimize}} \frac{\lambda}{m}\sum_{i=1}^{m}\|\sigma(W^T\sigma(Wx^{(i)}+b)+c) - x^{(i)}\|_2^2 + \sum_{i=1}^{m}\sum_{j=1}^{k} g(W_j x^{(i)})$$



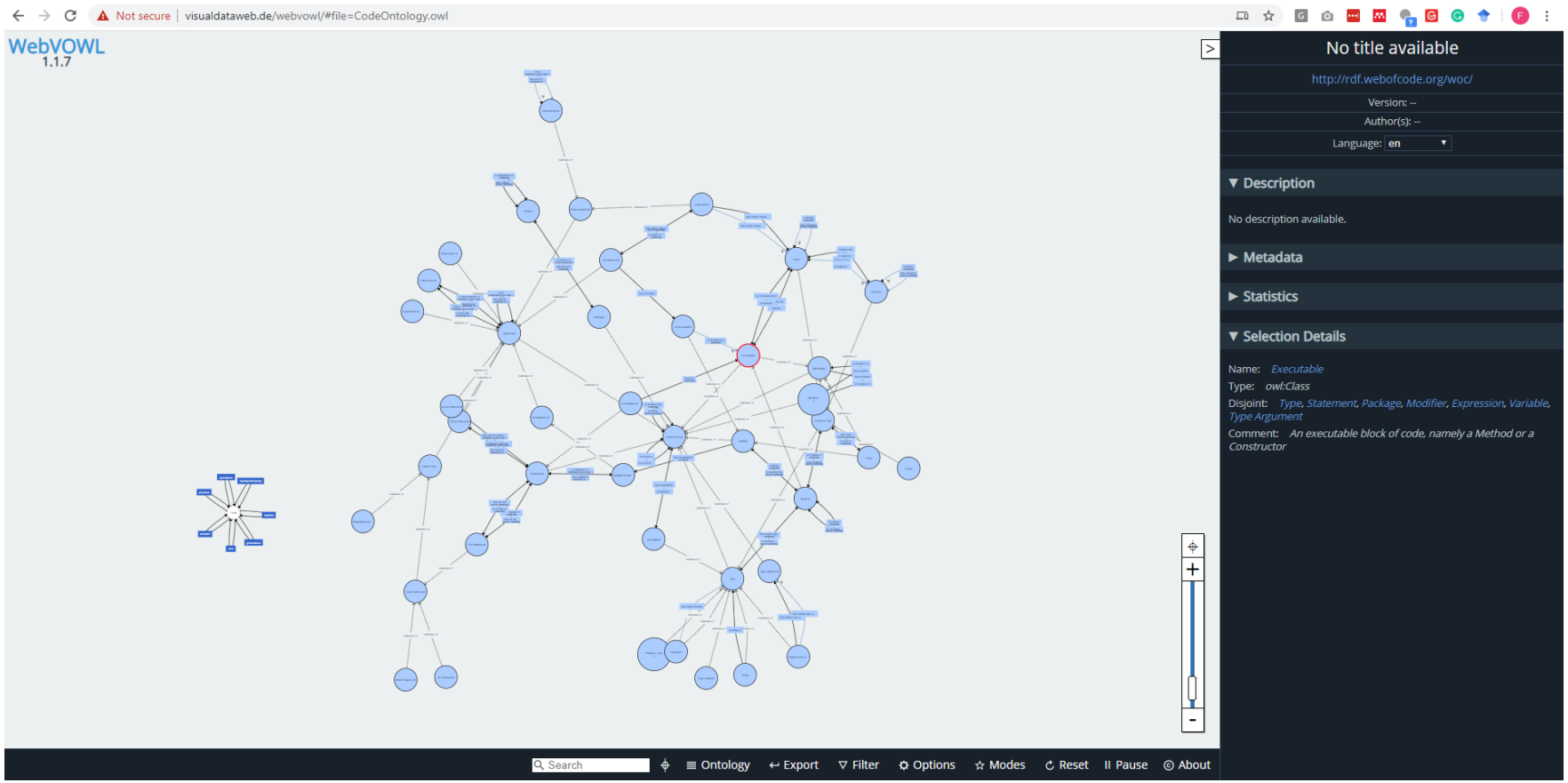


DEEPCODE

Theory Refresher – Part 3: Symbolic vs Sub-symbolic AI

Symbolic AI	Subsymbolic AI
Explicit symbolic programming Inference, search algorithms AI programming languages Rules, Ontologies, Plans, Goals	Bayesian learning Deep learning Connectionism Neural Nets/Backprop LDA, SVM, HMM, PMF, alphabet soup…
Introspection more useful for coding Easier to debug Easier to explain Easier to control Not so Big Data More useful for explaining people's thought Better for abstract problems	More robust against noise Better performance Less knowledge upfront Easier to scale up Big Data More useful for connecting to neuroscience Better for perceptual problems

Ontology



http://www.visualdataweb.de/webvowl/

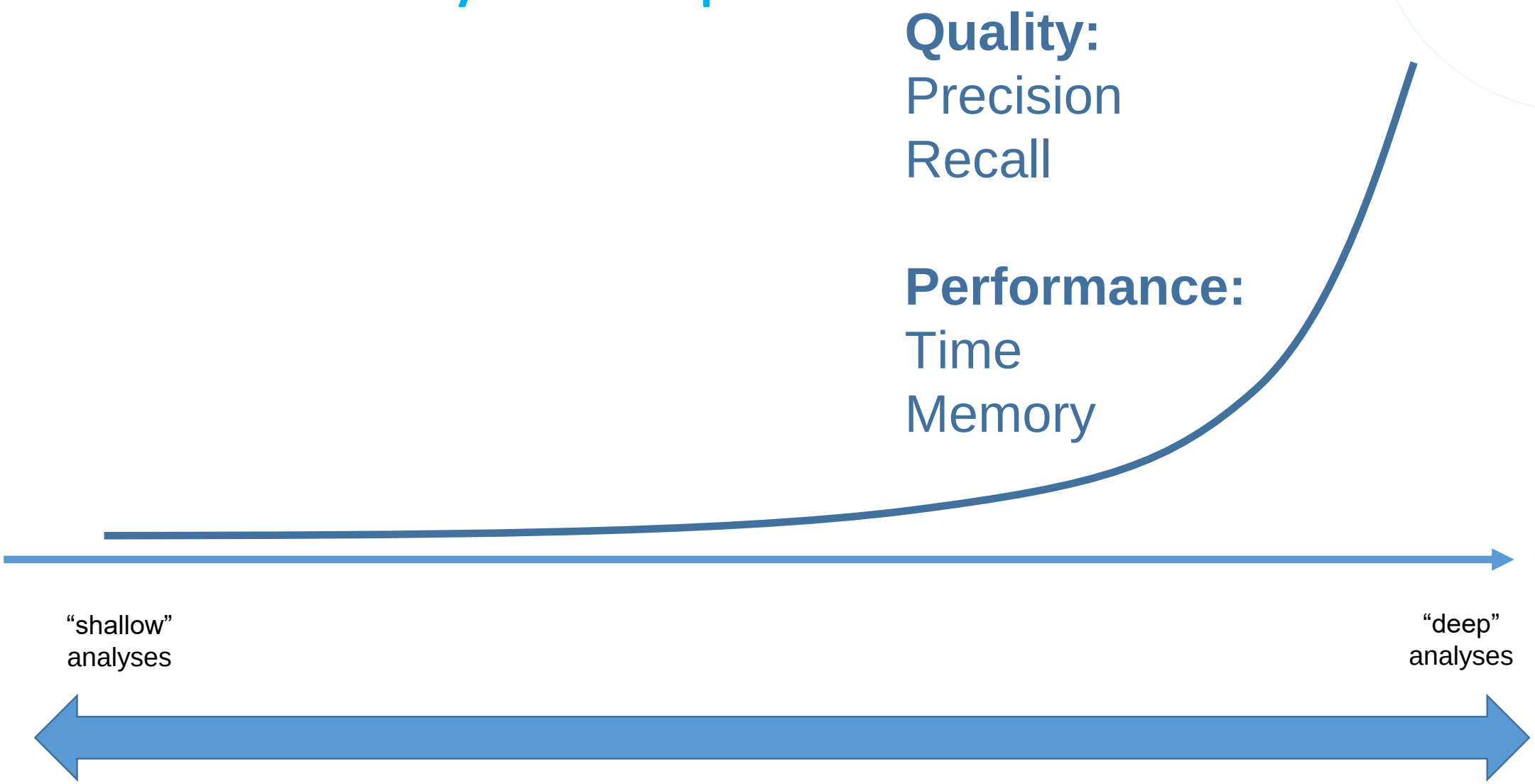
An **ontology** describes a structure for a body of knowledge. With the structure and similarities between ontologies, you can generalize knowledge.

DEEPCODE

Static Program Analysis

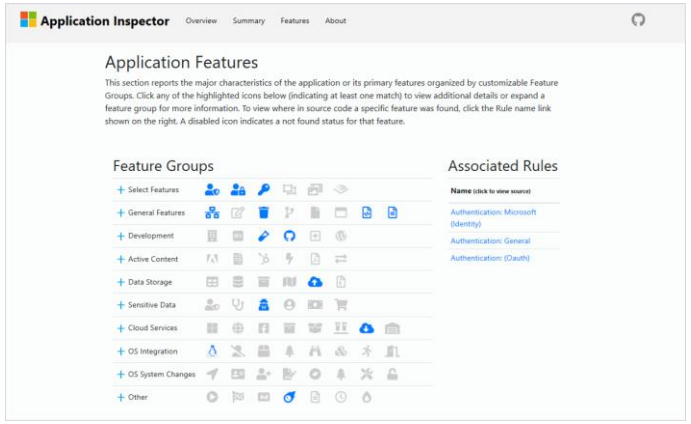
DEEPCODE

Spectrum of Static Analysis Techniques



DEEPCODE

Textual processing – the very basic tools and ML



Treats Source Code as Text

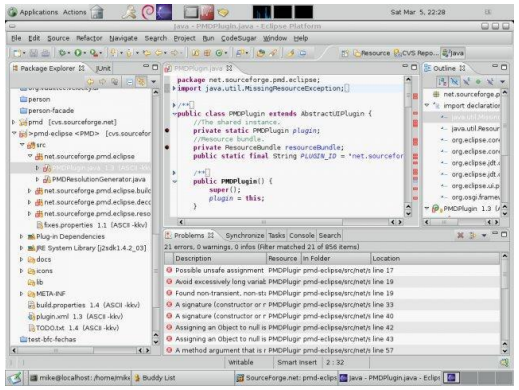
- Simple API misuse, deprecated APIs
- Leaked Secrets, Authentication Tokens

Examples

- Simplest grep
- Microsoft Application Inspector

DEEPCODE

Linters and AST



Constructs Abstract Syntax Tree and runs Rule Sets

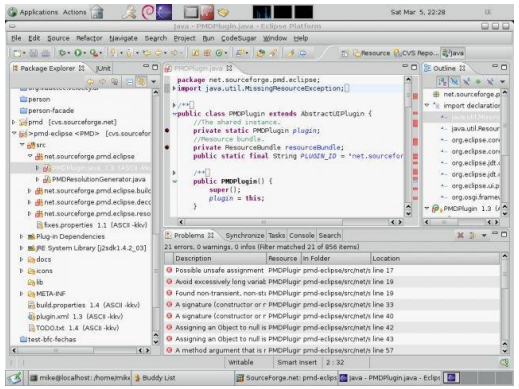
- Style violations
- Local Bug Patterns
- Mostly Language dependent

Examples

- PMD
- eslint

DEEPCODE

Linters and AST



Co

- S
- L
- L

Exa

- P

“UseEqualsToCompareStrings” Rule

If(com == “end”)

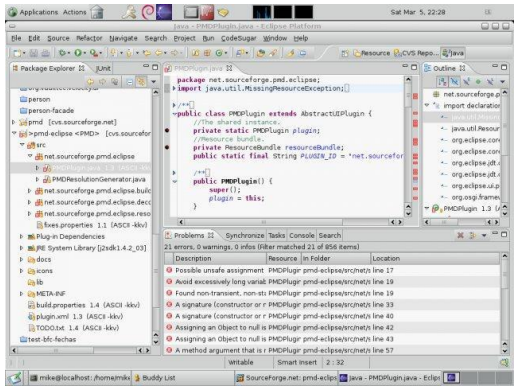
If(“end”.equals(com))

“shallow” analyses

“deep” analyses

DEEPCODE

Linters and AST



Co

- S
- L
- L

Exa

- P

“UseEqualsToCompareStrings” Rule

!

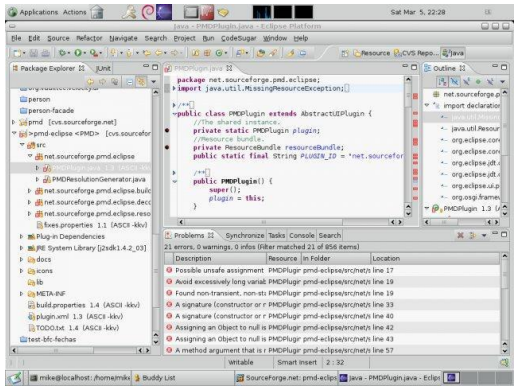
```
void equalsIgnoreCase(String x, String y) {
Return x.toLowerCase() == y.toLowerCase(); }
```

“shallow” analyses

“deep” analyses

DEEPCODE

Linters and AST -Medium



Constructs Abstract Syntax Tree and runs Rule Sets

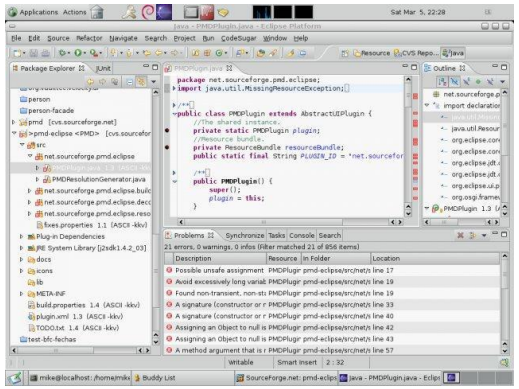
- Abstract Syntax Tree
- Symbol Table
- Resolve Names
- Type Propagation
- Dataflow Analysis

Examples

- Error Prone

DEEPCODE

Linters and AST - Medium



Co

- A
- S
- R
- T
- D

Exa

- E

“NullTernary” Rule

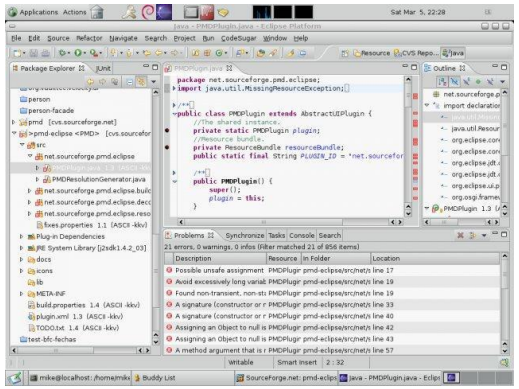
```
int x = flag ? foo : null;
```

“shallow” analyses

“deep” analyses

DEEPCODE

Linters and AST - Medium



Co

- A
- S
- R
- T
- D

Exa

- E

“NullTernary” Rule

!

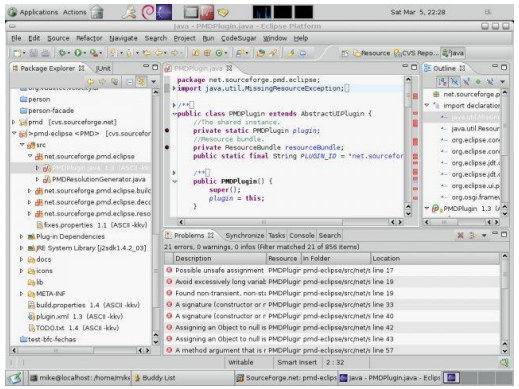
```
Integer y = null;
int x = flag ? foo : y;
```

“shallow” analyses

“deep” analyses

DEEPCODE

Linters and AST - Medium



Co

- A
- S
- R
- T
- D

Exa

- E

“NullTernary” Rule

!

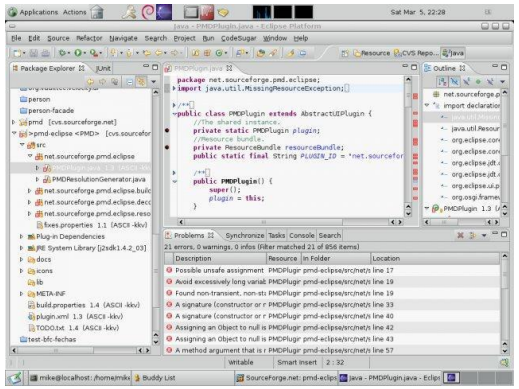
```
Integer y = someAPIFnWithNull();
int x = flag ? foo : y;
```

“shallow” analyses

“deep” analyses

DEEPCODE

Intermediate Representations

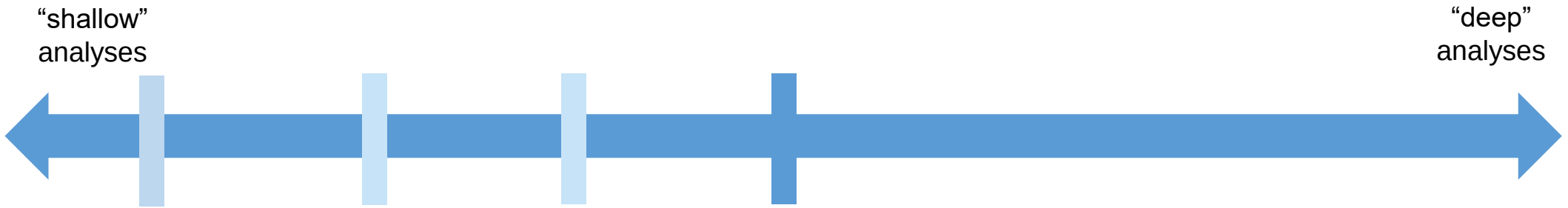


Constructs Abstract Syntax Tree and runs Rule Sets

- Interprocedural (generate context for each procedure)

Examples

- fbinfer



DEEPCODE

Infer Example

```
864.
865. public void feedItemSelected(String feedId) {
866. FeedObject feedObject = DDGApplication.getDB().selectFeedById(feedId);
867. feedItemSelected(feedObject);
```

```
482.
483. public FeedObject selectFeedById(String id){
484. FeedObject out = null;
485. Cursor c = null;
486. try {
487.   c = this.db.query(FEED_TABLE, null, "_id=?", new String[]{id}, null, null, null);
488.   if (c.moveToFirst()) {
489.    out = getFeedObject(c);
490.   }
491.  } finally {
492.      if(c!=null) {
493.           c.close();
494.      }
495.  }
496.  return out;
497. }
```

```
841.
842.  public void feedItemSelected(FeedObject feedObject) {
843. // keep a reference, so that we can reuse details while saving
844.    DDGControlVar.currentFeedObject = feedObject;
845.    DDGControlVar.mDuckDuckGoContainer.sessionType = SESSIONTYPE.SESSION_FEED;
846.
847     String url = feedObject.getUrl();
```

DEEPCODE

Towards Verification – Using Annotations

```
static void log(Object x) {
        System.out.println(x.toString());
        }

static void foo() {
        log(null);
        }
```

- Uber's NullAway
  (Data Flow Analysis / Checker Framework)

```
static void log(@NotNull Object x) {
        System.out.println(x.toString());
        }

static void foo() {
        log(null);
        }
```

-> Annotations unlock the Semantic

-> Annotations are expensive ☹

DEEPCODE

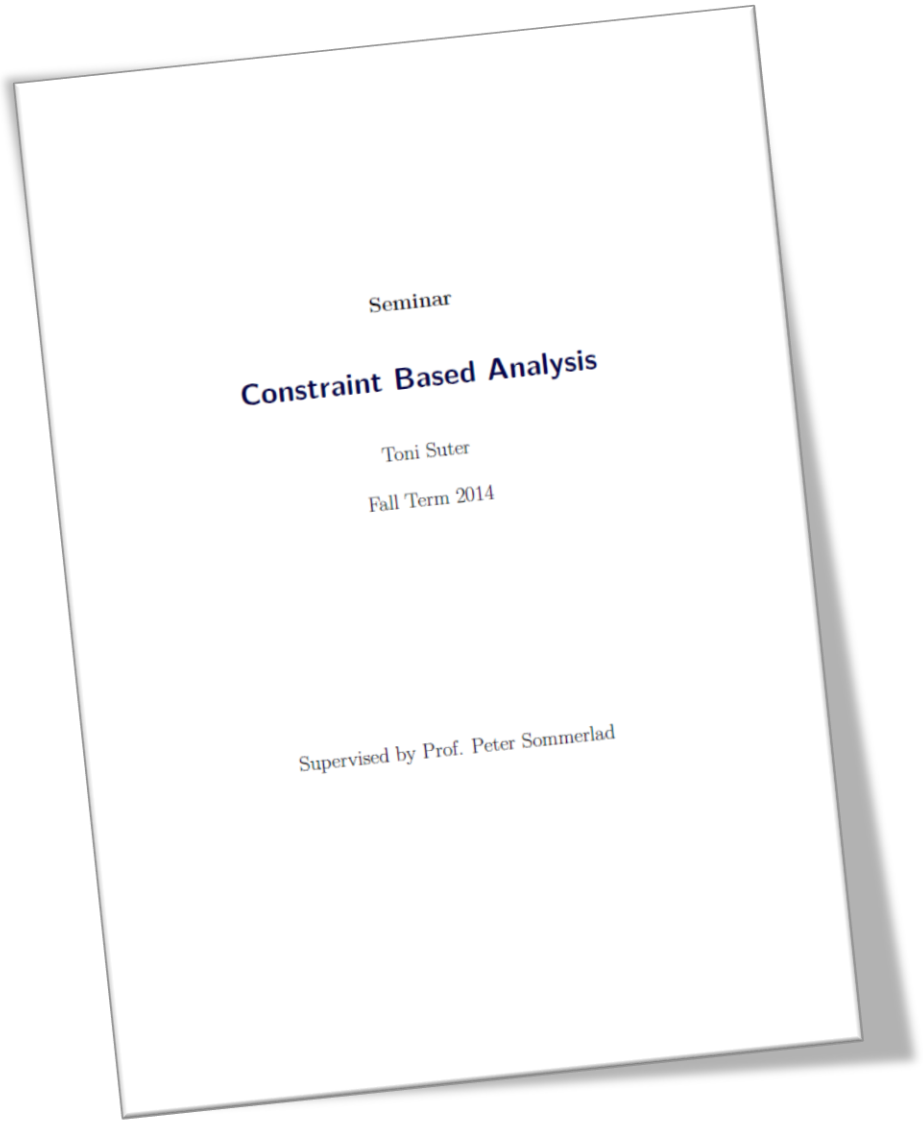
Overview Offerings

- Sonarqube
- Coverty
- Cobertura
- Snyk (Security)
- Checkmarx (Security)
- IntelliJ
- DiffBlue
- …

- Findbugs
- PMD
- Checkstyle
- Infer
- NullAway
- Semmle
- EMMA (Code coverage)
- …

DEEPCODE

How do these tools work? Constraint Based Analysis



Seminar

Constraint Based Analysis

Toni Suter

Fall Term 2014

Supervised by Prof. Peter Sommerlad

- Build an intermediate representation (e.g., a graph)
- Use logic programing to build knowledge about the graph
- Use constraints („rules“) on the knowledge (e.g. types need to match)

- See: https://wiki.ifs.hsr.ch/SemProgAnTr/files/ConstraintBasedAnalysis.pdf

DEEPCODE

Today's Limitations

Performance

Inter-procedural or inter-file is extremely resource intense

Quality

Recall versus Precision, types of bugs

Polyglot

Mix of languages, tools need to adapt

Linear Scaling of Rules

Handcrafting rules by experts scales linearly at best

Expressiveness

Found issues need to be explanatory and actionable

API Use

Ever changing libraries (if they would be annotated ☺)

DEEPCODE

DeepCode's Approach

Performance

Inter-procedural or inter-file is extremely resource intense

Inhouse build constraint solver using Datalog
Advanced Data Structures

DEEPCODE

DeepCode's Approach

Semi-Supervised Learning using Big Code

Quality

Recall versus

Polyglot

Mix of languages, tools need to adapt

ing of Rules

ng rules by experts scales
best

ness

es need to be explanatory
actionable

API Use

Ever changing libraries (if they would be annotated ☺)

DEEPCODE

DeepCode's Approach

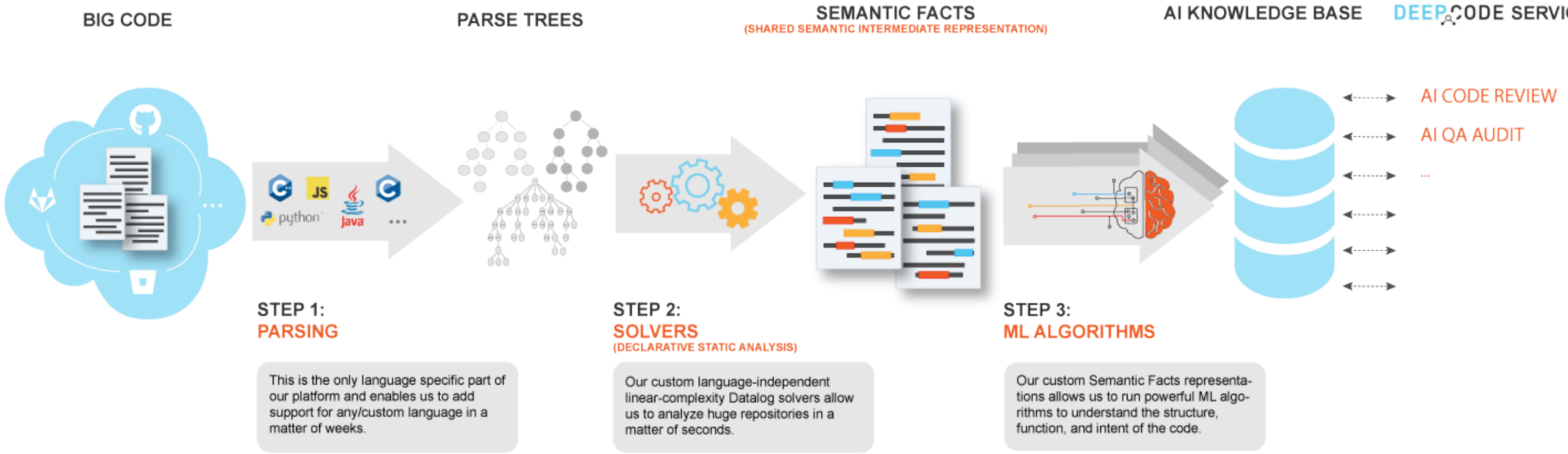
Ontologies

Expressiveness

Found issues need to be explanatory and actionable

DEEPCODE

Event Graphs and Symbolic AI and Ontologies



“shallow” analyses

“deep” analyses

DEEPCODE

Demo

- OWASP/Benchmark (https://github.com/OWASP/Benchmark)
- JFX (https://github.com/openjdk/jfx)
- JDK14 (https://github.com/openjdk/jdk14)

DEEPCODE

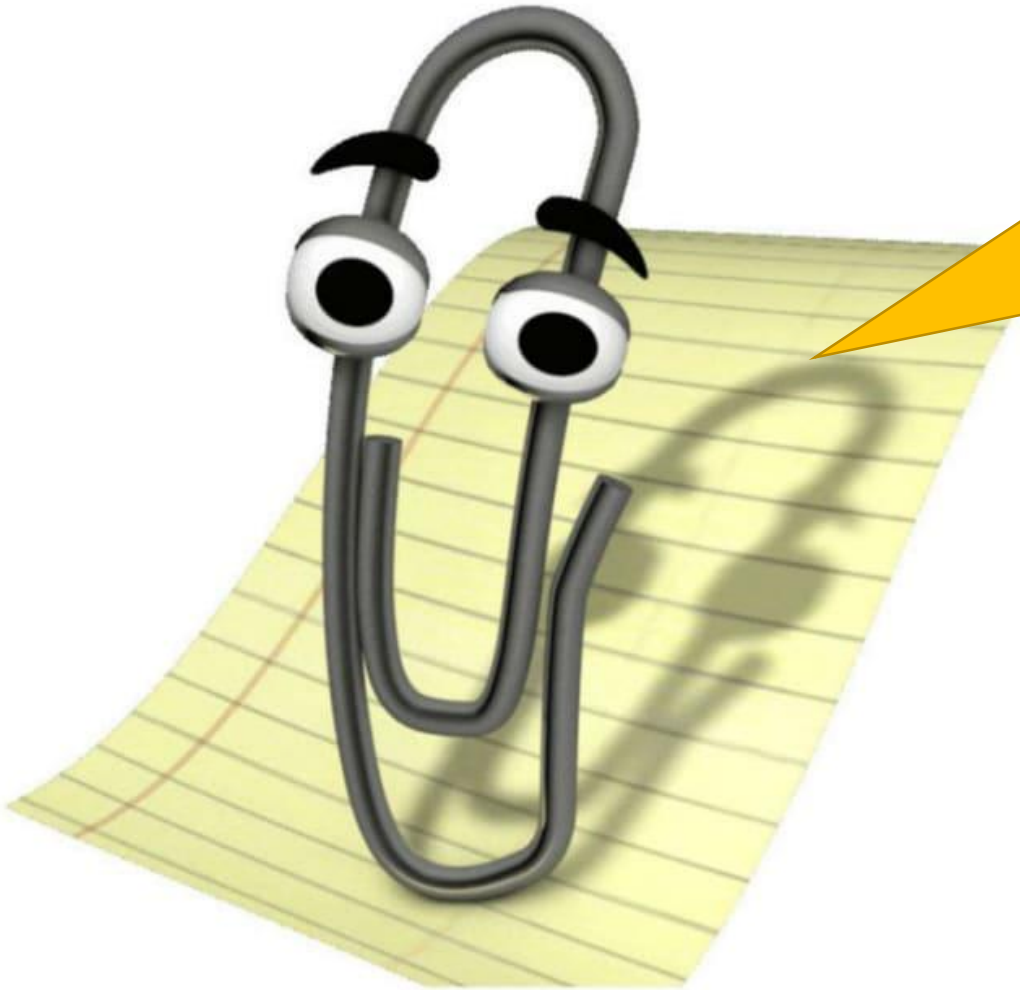
Dos and Don'ts – how to build your tool stack

- Success measure should be: Bug fixed not issues identified
- Shift left is good – but too much left hurts*
  - Tools can break the development process (compiler errors)
  - But developers are more likely to fix bugs when presented early
- The sieve gets finer to the right
  - Apply Cheese Slice Model
- Integration into development process and developer workflow is key*
  - IDE (Style+Tool Integration)
  - Compiler
  - Continuous Integration
- Use tools on legacy code bases
  - Revisit legacy code (like 6 months, update libraries, run CI, maybe CD)

* According to Google ☺ https://cacm.acm.org/magazines/2018/4/226371-lessons-from-building-static-analysis-tools-at-google/fulltext

DEEPCODE

Outlook – what to expect more

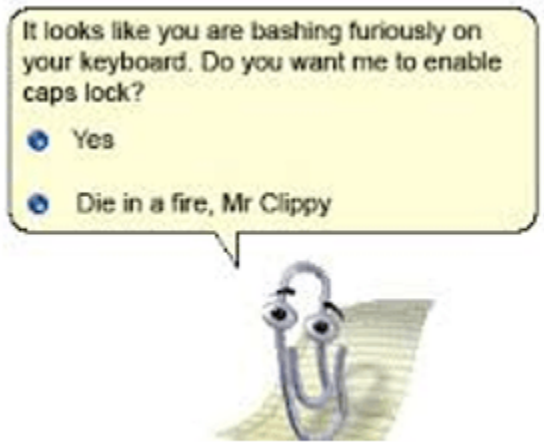


It seems you want to develop 3D First Person Shooter for Arduino. Shall I help you with that?

DEEPCODE

Outlook – what to expect more

- Generalization drives Polyglot
- DevOps
- Helpful
  - Semantic: Reasoning of likely meaning/intention
  - Expressiveness:
    - Explain the reasons
    - Find alternatives to current construction
  - Find examples for unit tests, generate unit tests
- Rules scale exponential by using ML
- Combination of Static and Dynamic
- Combination of Static and Dynamic and Runtime



DEEPCODE

Call to Action

- If you use Static Program Analysis…
  - Bravo!
  - Revisit your toolstack and optimize
- If you are not using Static Program Analysis…
  - It is ok, you are amongst friends
  - Look into your projects, try some tools and see what fits you the most
  - IntelliJ is a good start

Give us a try ☺ -> **deepcode.ai** <-

DEEPCODE

Thank you…

DEEPCODE