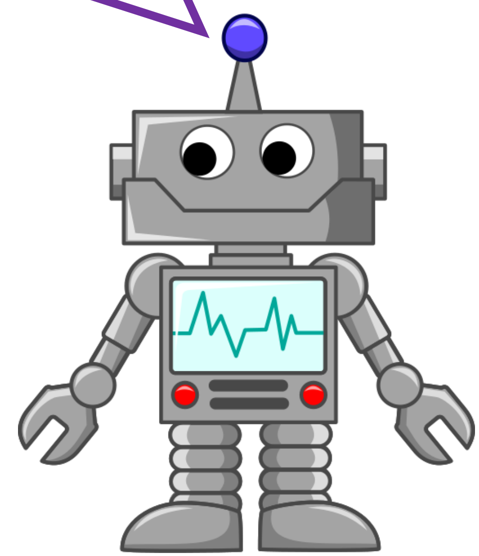
Natural language is a programming language

Michael D. Ernst

UW CSE



Joint work with Arianna Blasi, Juan Caballero, Sergio Delgado Castellanos, Alberto Goffi, Alessandra Gorla, Victoria Lin, Deric Pang, Mauro Pezzè, Irfan Ul Haq, Kevin Vu, Luke Zettlemoyer, and Sai Zhang

Questions about software

- How many of you have used software?
- How many of you have written software?

What is software?

What is software?

- A sequence of instructions that perform some task

What is software?

An engineered object amenable to formal analysis

- A sequence of instructions that perform some task

Formalizations

$\mathtt{h}$	$\in$	$\mathtt{Heap}$	$=$	$\mathtt{Addr} \rightharpoonup \mathtt{Obj}$
ι	$\in$	$\mathtt{Addr}$	$=$	Set of Addresses $\cup \{\mathtt{null}_a\}$
$\mathtt{o}$	$\in$	$\mathtt{Obj}$	$=$	${}^{r}\mathtt{Type},\ \mathtt{Fields}$
${}^{r}\mathtt{T}$	$\in$	${}^{r}\mathtt{Type}$	$=$	$\mathtt{OwnerAddr\ ClassId}\texttt{<}\overline{{}^{r}\mathtt{Type}}\texttt{>}$
$\mathtt{Fs}$	$\in$	$\mathtt{Fields}$	$=$	$\mathtt{FieldId} \rightharpoonup \mathtt{Addr}$
ι	$\in$	$\mathtt{OwnerAddr}$	$=$	$\mathtt{Addr} \cup \{\mathtt{any}_a\}$
${}^{r}\Gamma$	$\in$	${}^{r}\mathtt{Env}$	$=$	$\overline{\mathtt{TVarId}\ {}^{r}\mathtt{Type};}\ \overline{\mathtt{ParId\ Addr}}$

$\mathtt{P}$	$\in$	$\mathtt{Program}$	$::=$	$\overline{\mathtt{Class}},\ \mathtt{ClassId},\ \mathtt{Expr}$
$\mathtt{Cls}$	$\in$	$\mathtt{Class}$	$::=$	$\mathtt{class\ ClassId}\texttt{<}\overline{\mathtt{TVarI}}$ … $\mathtt{extends\ ClassId}\texttt{<}\overline{{}^{s}\mathtt{Ty}}$ … $\{\ \overline{\mathtt{FieldId}\ {}^{s}\mathtt{Type};}\ \mathtt{Met}$ …
${}^{s}\mathtt{T}$	$\in$	${}^{s}\mathtt{Type}$	$::=$	${}^{s}\mathtt{NType} \mid \mathtt{TVarId}$
${}^{s}\mathtt{N}$	$\in$	${}^{s}\mathtt{NType}$	$::=$	$\mathtt{OM\ ClassId}\texttt{<}\overline{{}^{s}\mathtt{Type}}\texttt{>}$
$\mathtt{u}$	$\in$	$\mathtt{OM}$	$::=$	
$\mathtt{mt}$	$\in$	$\mathtt{Meth}$	$::=$	
		$\mathtt{MethSig}$	$::=$	
$\mathtt{w}$	$\in$	$\mathtt{Purity}$	$::=$	
$\mathtt{e}$	$\in$	$\mathtt{Expr}$	$::=$	$\mathtt{Expr.MethId}\texttt{<}{}^{s}\mathtt{Type}\texttt{>}(\mathtt{Expr}) \mid \mathtt{new}\ {}^{s}\mathtt{Type} \mid ({}^{s}\mathtt{Type})\ \mathtt{Expr}$
${}^{s}\Gamma$	$\in$	${}^{s}\mathtt{Env}$	$::=$	$\overline{\mathtt{TVarId}\ {}^{s}\mathtt{NType};}\ \overline{\mathtt{ParId}\ {}^{s}\mathtt{Type}}$

$$\text{OS-Upd}\ \frac{\begin{array}{c}\mathtt{h}, {}^{r}\Gamma, \mathtt{e}_0 \rightsquigarrow \mathtt{h}_0, \iota_0 \\ \iota_0 \neq \mathtt{null}_a \\ \mathtt{h}_0, {}^{r}\Gamma, \mathtt{e}_2 \rightsquigarrow \mathtt{h}_2, \iota \\ \mathtt{h}' = \mathtt{h}_2[\iota_0.\mathtt{f} := \iota]\end{array}}{\mathtt{h}, {}^{r}\Gamma, \mathtt{e}_0.\mathtt{f}{=}\mathtt{e}_2 \rightsquigarrow \mathtt{h}',}$$

$$\text{OS-Read}\ \frac{\begin{array}{c}\mathtt{h}, {}^{r}\Gamma, \mathtt{e}_0 \rightsquigarrow \mathtt{h}', \iota_0 \\ \iota_0 \neq \mathtt{null}_a \\ \iota = \mathtt{h}'(\iota_0)\downarrow_2(\mathtt{f})\end{array}}{\mathtt{h}, {}^{r}\Gamma, \mathtt{e}_0.\mathtt{f} \rightsquigarrow \mathtt{h}', \iota}$$

$$\text{GT-Read}\ \frac{\Gamma \vdash \mathtt{e}_0 : \mathtt{N}_0 \qquad \mathtt{N}_0 = \_\ (}{\Gamma \vdash \mathtt{e}_0.\mathtt{f} : \mathtt{N}_0 \triangleright fType(\mathtt{C}_0, \mathtt{f})}$$

$$\text{GT-Upd}\ \frac{\begin{array}{c}\Gamma \vdash \mathtt{e}_0 : \mathtt{N}_0 \qquad \mathtt{N}_0 = \mathtt{u}_0\ \mathtt{C}_0\texttt{<\_>} \\ \mathtt{T}_1 = fType(\mathtt{C}_0, \mathtt{f}) \\ \Gamma \vdash \mathtt{e}_2 : \mathtt{N}_0 \triangleright \mathtt{T}_1 \\ \mathtt{u}_0 \neq \mathtt{any} \qquad rp(\mathtt{u}_0, \mathtt{T}_1)\end{array}}{\Gamma \vdash \mathtt{e}_0.\mathtt{f}{=}\mathtt{e}_2 : \mathtt{N}_0 \triangleright \mathtt{T}_1}$$

$$\left.\begin{array}{l}\mathtt{h} \vdash {}^{r}\Gamma : {}^{s}\Gamma \\ \mathtt{h} \vdash \iota_1 : dyn({}^{s}\mathtt{N}, \mathtt{h}, {}^{r}\Gamma) \\ \mathtt{h} \vdash \iota_2 : dyn({}^{s}\mathtt{T}, \iota_1, \mathtt{h}(\iota_1)\downarrow_1) \\ {}^{s}\mathtt{N} = \mathtt{u}_N\ \mathtt{C}_N\texttt{<>} \\ \mathtt{u}_N = \mathtt{this}_u \Rightarrow {}^{r}\Gamma(\mathtt{this}) \\ free({}^{s}\mathtt{T}) \subseteq dom(\mathtt{C}_N)\end{array}\right\} \implies \mathtt{h} \vdash \iota_2 : dyn({}^{s}\mathtt{N} \triangleright {}^{s}\mathtt{T}, \mathtt{h}, {}^{r}\Gamma)$$

$$\text{DYN}\ \frac{\begin{array}{ccc}{}^{r}\mathtt{T} = \iota'\ \_\texttt{<>} & \iota \vdash {}^{r}\mathtt{T}\ {}^{r}\!\!<:\ \iota'\ \mathtt{C}\texttt{<}\overline{{}^{r}\mathtt{T}}\texttt{>} & \iota \vdash {}^{r}\mathtt{T}\ {}^{r}\!\!<:\ \iota'\ \mathtt{C}\texttt{<}\overline{{}^{r}\mathtt{T}_a}\texttt{>} \Rightarrow \iota \vdash \overline{{}^{r}\mathtt{T}}\ {}^{r}\!\!<:\ \overline{{}^{r}\mathtt{T}_a} \\ \mathtt{dom}(\mathtt{C}) = \overline{\mathtt{X}} & & \mathtt{free}({}^{s}\mathtt{T}) \subseteq \overline{\mathtt{X}} \circ \overline{\mathtt{X}'}\end{array}}{dyn({}^{s}\mathtt{T}, \iota, {}^{r}\mathtt{T}, (\overline{\mathtt{X}'\ {}^{r}\mathtt{T}'}; \_)) = {}^{s}\mathtt{T}[\iota'/\mathtt{this}, \iota'/\mathtt{peer}, \iota/\mathtt{rep}, \mathtt{any}_a/\mathtt{any}_u, \overline{{}^{r}\mathtt{T}/\mathtt{X}}, \overline{{}^{r}\mathtt{T}'/\mathtt{X}'}]}$$

What is software?

- A sequence of instructions that perform some task

What is software?

- A sequence of instructions that perform some task

- Test cases
- Version control history
- Issue tracker
- Documentation
- …

How should it be analyzed?

Analysis of a natural object

- Machine learning over executions
- Version control history analysis
- Bug prediction
- Upgrade safety
- Prioritizing warnings
- Program repair

Specifications are needed; Tests are available but ignored

- Many papers start:
  "Given a program and its specification…"
- Formal verification process:
  - Write the program
  - Test the program
  - Verify the program, *ignoring* testing artifacts

Programmers embed semantic info in tests

Goal: translate tests into specifications,
by machine learning over executions

Dynamic detection of likely invariants

https://plse.cs.washington.edu/daikon/
[ICSE 1999]

- Observe values that the program computes
- Generalize over them via machine learning
- Result: invariants (as in **`assert`**s or specifications)
  - **`x > abs(y)`**
  - **`x = 16*y + 4*z + 3`**
  - array **`a`** contains no duplicates
  - for each node **`n`**, **`n=n.child.parent`**
  - graph **`g`** is acyclic
- Unsound, incomplete, and useful

Applying NLP to software engineering

Problems		NL sources	NLP techniques
inadequate diagnostics	Analyze existing code	error messages	document similarity
incorrect operations		variable names	word semantics
missing tests	Generate new code	code comments	parse trees
unimplemented functionality		user questions	translation

Applying NLP to software engineering

Problems	NL sources	NLP techniques
inadequate diagnostics	error messages	document similarity
incorrect operations	variable names	word semantics
missing tests	code comments	parse trees
unimplemented functionality	user questions	translation

[ISSTA 2015]

Inadequate diagnostic messages

Scenario: user supplies a wrong configuration option

`--port_num=100.0`

Problem: software issues an unhelpful error message

- "unexpected system failure"
- "unable to establish connection"

Hard for end users to diagnose

Goal: detect such problems *before* shipping the code

- Better message: "`--port_num` should be an integer"

Challenges for proactive detection of inadequate diagnostic messages

- **How to *trigger a configuration error*?**

- **How to *determine the inadequacy* of a diagnostic message?**

ConfDiagDetector's solutions

- **How to *trigger a configuration error*?**
  - Configuration mutation + run system tests

configuration

+

system tests

failed tests ≈ triggered errors

(We know the root cause.)

- **How to *determine the inadequacy* of a diagnostic message?**
  - Use a NLP technique to check its semantic meaning

Similar semantic meanings?

Diagnostic messages output by failed tests

User manual
(Assumption: a manual, webpage, or man page exists.)

When is a message adequate?

- Contains the mutated **option name** or value [Keller'08, Yin'11]

  Mutated option:

  `--percentage-split`

  Diagnostic message:

  "**`the value of percentage-split should be > 0`**"

- Similar **semantic meaning** as the manual description

  Mutated option:

  `--fnum`

  Diagnostic message:

  "**`Number of folds must be greater than 1`**"

  User manual description of **`--fnum`**:

  "**`Sets number of folds for cross-validation`**"

Classical document similarity: TF-IDF + cosine similarity

1. Convert document into a real-valued vector
2. Document similarity = vector cosine similarity

- Vector length = dictionary size, values = term frequency (TF)
  - Example: [2 $_{\text{classical}}$, 8 $_{\text{document}}$, 3 $_{\text{problem}}$, 3 $_{\text{values}}$, …]
- Problem: frequent words swamp important words
- Solution: values = TF x IDF (inverse document frequency)
  - IDF = log(total documents / documents with the term)

Problem: does not work well on very short documents

Text similarity technique [Mihalcea'06]

A message

Manual description

The documents have similar semantic meanings
if many words in them have similar meanings

Example:

~~The~~ program goes wrong

~~The~~ software fails

1. Remove all stop words
2. For each word in the diagnostic message, try to find similar words in the manual
3. Two sentences are similar, if "many" words are similar between them.

Results

- Reported 25 missing and 18 inadequate messages in Weka, JMeter, Jetty, Derby
- Validation by 3 programmers:
  - 0% false negative rate
    - Tool says message is adequate, humans say it is inadequate
  - 2% false positive rate
    - Tool says message is inadequate, humans say it is adequate
    - Previous best: 16%

Related work

Configuration error diagnosis techniques

- Dynamic tainting [Attariyan'08], static tainting [Rabkin'11], Chronus [Whitaker'04]

Troubleshooting an exhibited error rather than detecting inadequate diagnostic messages

Software diagnosability improvement techniques

- PeerPressure [Wang'04], RangeFixer [Xiong'12], ConfErr [Keller'08] and Spex-INJ [Yin'11], EnCore [Zhang'14]

Requires source code, usage history, or OS-level support

Applying NLP to software engineering

Problems	NL sources	NLP techniques
inadequate diagnostics	error messages	document similarity
incorrect operations	variable names	word semantics
missing tests	code comments	parse trees
unimplemented functionality	user questions	translation

[WODA 2015]

Undesired variable interactions

```
int totalPrice;
int itemPrice;
int shippingDistance;

totalPrice = itemPrice + shippingDistance;
```

Undesired variable interactions

```
int totalPrice;
int itemPrice;
int shippingDistance;
totalPrice = itemPrice + shippingDistance;
```

- The compiler issues no warning
- A human can tell the abstract types are different

Idea:

- Cluster variables based on words in variable names
- Cluster variables based on usage in program operations

Differences indicate bugs or poor variable names

Undesired interactions

`distance`	`itemPrice`	`tax_rate`
`miles`	`shippingFee`	`percent_complete`

Undesired interactions

`distance` ↔ `itemPrice` `tax_rate`

`itemPrice + distance`

`miles` `shippingFee` `percent_complete`

Undesired interactions

`int`

`distance` ↔ `itemPrice`

`miles` `shippingFee`

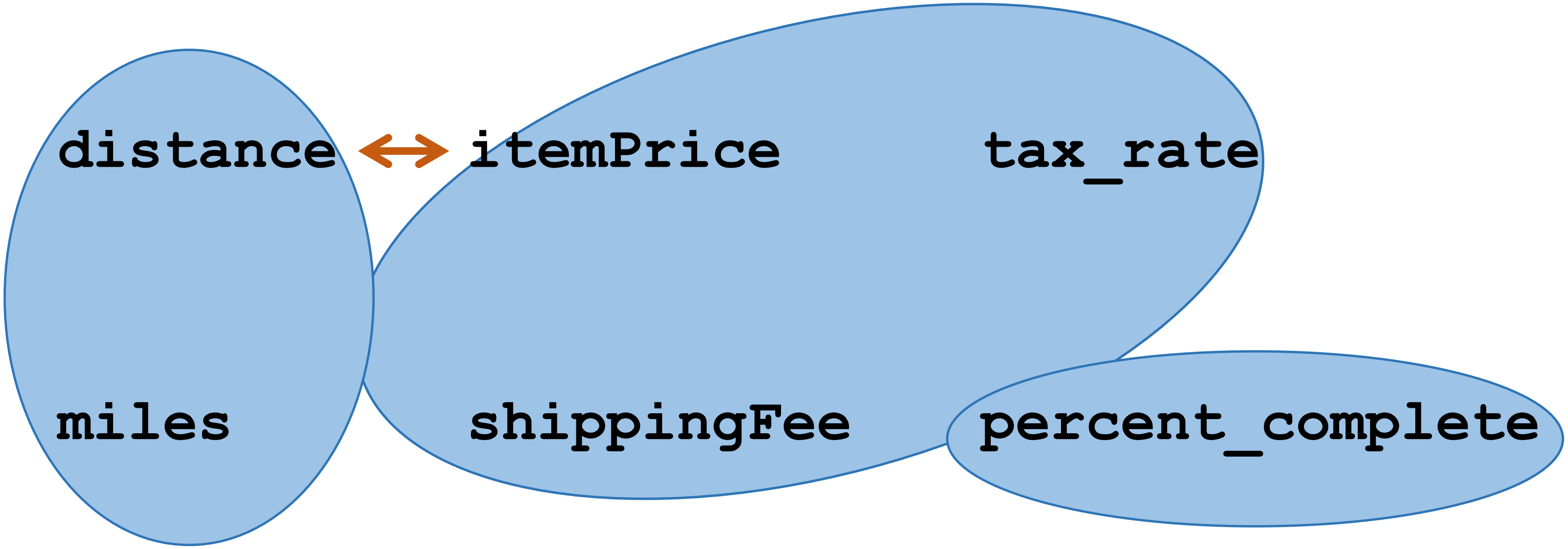
`float`

`tax_rate`

`percent_complete`

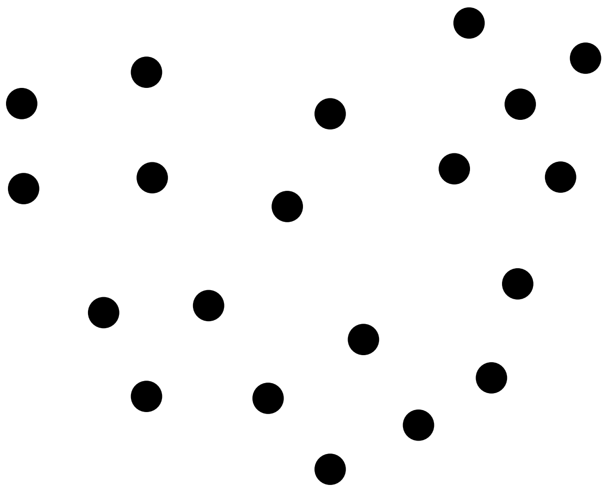
Program types don't help

Undesired interactions



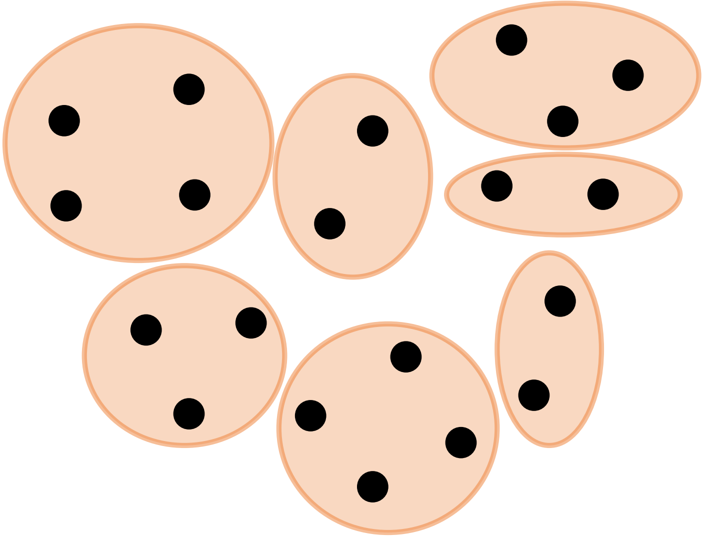
Language indicates the problem

Variables



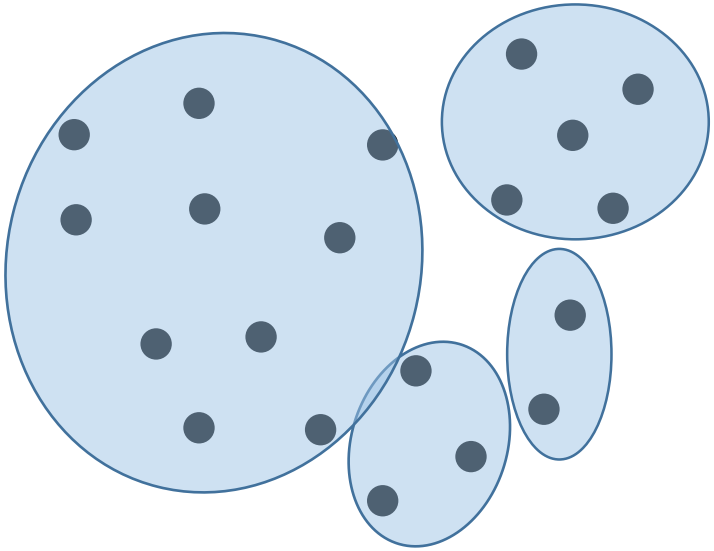
Variable clustering

Cluster based on interactions: operations



Variable clustering

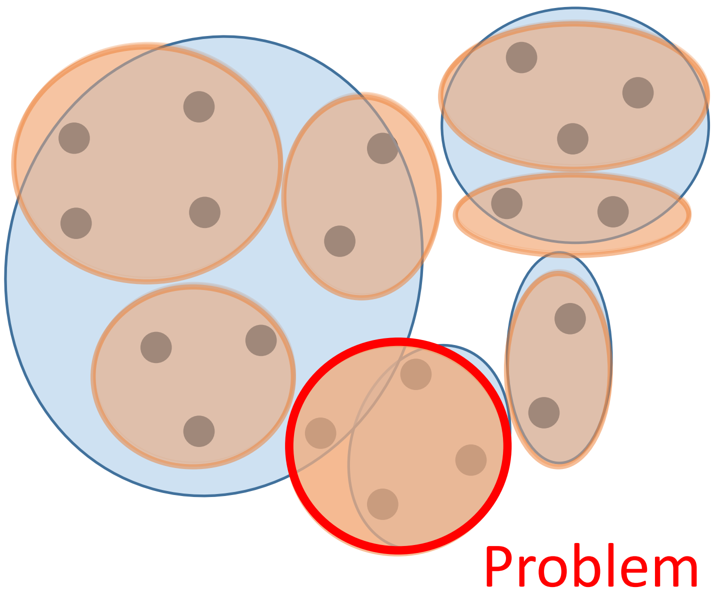
Cluster based on
language:
variable names



Variable clustering

Cluster based on interactions: operations

Cluster based on language: variable names



Actual algorithm:

1. Cluster based on operations
2. Sub-cluster based on names
3. Rank an operation cluster as suspicious if it contains well-defined name sub-clusters

Clustering based on operations

Abstract type inference [ISSTA 2006]

```
int totalCost(int miles, int price, int tax) {
  int year = 2016;
  if ((miles > 1000) && (year > 2000)) {
    int shippingFee = 10;
    return price + tax + shippingFee;
  } else {
    return price + tax;
  }
}
```

Clustering based on operations

Abstract type inference [ISSTA 2006]

```
int totalCost(int miles, int price, int tax) {
   int year = 2016;
   if ((miles > 1000) && (year > 2000)) {
      int shippingFee = 10;
      return price + tax + shippingFee;
   } else {
      return price + tax;
   }
}
```

Clustering based on variable names

Compute variable name similarity for var_1 and var_2

1. Tokenize each variable into dictionary words
   - **`in_authskey15`** ⇒ {"in", "authentications", "key"}
   - Expand abbreviations, best-effort tokenization
2. Compute word similarity
   - For all $w_1 \in var_1$ and $w_2 \in var_2$, use WordNet (or edit distance)
3. Combine word similarity into variable name similarity
   - $\text{maxwordsim}(w_1, var_2) = \max_{w_2 \in var_2} \text{wordsim}(w_1, w_2)$
   - $\text{varsim}(var_1, var_2) = \underset{w1 \in var1}{\text{average}} \ \text{maxwordsim}(w_1, var_2)$

Results

- Ran on grep and Exim mail server
- Top-ranked mismatch indicates
  an undesired variable interaction in grep

```
if (depth < delta[tree->label])
    delta[tree->label] = depth;
```

- Loses top 3 bytes of depth
- Not exploitable because of guards elsewhere in program, but not obvious here

Related work

- Reusing identifier names is error-prone [Lawrie 2007, Deissenboeck 2010, Arnaoudova 2010]
- Identifier naming conventions [Simonyi]
- Units of measure [Ada, F#, etc.]
- Tokenization of variable names [Lawrie 2010, Guerrouj 2012]

Applying NLP to software engineering

Problems	NL sources	NLP techniques
inadequate diagnostics	error messages	document similarity
incorrect operations	variable names	word semantics
missing tests	code comments	parse trees
unimplemented functionality	user questions	translation

[ISSTA 2016]

Test oracles (`assert` statements)

A test consists of

- an input (for a unit test, a sequence of calls)
- an oracle (an `assert` statement)

Programmer-written tests

- often trivial oracles, or too few tests

Automatic generation of tests:

- inputs are easy to generate
- oracles remain an open challenge

Goal: create test oracles
from what programmers already write

Automatic test generation

- Code under test:

```
public class FilterIterator implements Iterator {
  public FilterIterator(Iterator i, Predicate p) {…}
  public Object next() {…}
  …
}
```

```
/** @throws NullPointerException if either
 * the iterator or predicate are null */
```

- Automatically generated test:

```
public void test {
  FilterIterator i = new FilterIterator(null, null);
  i.next();
}
```

Throws NullPointerException!

Did the tool discover a bug?

It could be:

1. Expected behavior
2. Illegal input
3. Implementation bug

Automatically generated tests

- A test generation tool outputs:
  - Passing tests – useful for regression testing
  - Failing tests – indicates a program bug
- Without a specification, the tool **guesses** whether a given behavior is **correct**
  - <u>False positives</u>: report a failing test that was due to illegal inputs
  - <u>False negatives</u>: fail to report a failing test because it might have been due to illegal inputs
- Results: Reduced false positive test failures in EvoSuite by 1/3 or more

Programmers write code comments

Javadoc is standard procedure documentation

```
/**
 * Checks whether the comparator is now
 * locked against further changes.
 *
 * @throws UnsupportedOperationException
 * if the comparator is locked
 */
protected void checkLocked() {...}
```

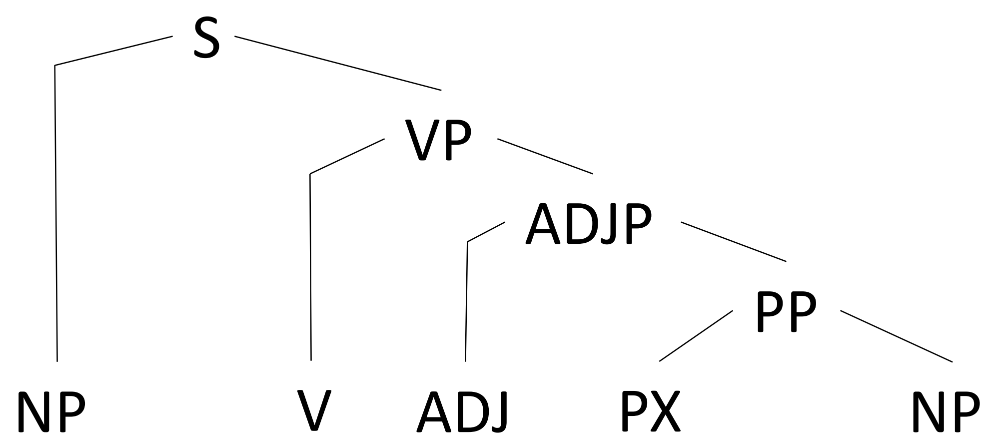
Javadoc comment and assertion

```
class MyClass {

  ArrayList allFoundSoFar = …;

  boolean canConvert(Object arg) { … }

  /** @throws IllegalArgumentException if the
   *  element is not in the list and is not
   *  convertible.  */
  void myMethod(Object element) { … }
}
```

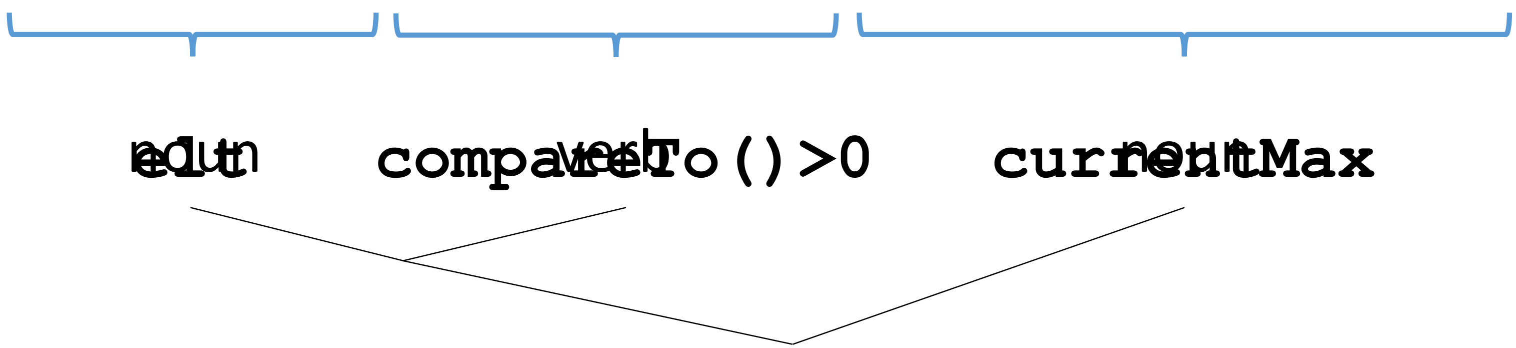
Condition for exception: **`myMethod`** should throw iff …

```
  ( !allFoundSoFar.contains(element)
    && !canConvert(element) )
```

Nouns = objects, verbs = operations



The element is greater than the current maximum.



`elt.compareTo(currentMax) > 0`

Text to code: Toradocu algorithm

1. Parse `@param`, `@return`, and `@throws` expressions using the Stanford Parser
   - Parse tree, grammatical relations, cross-references
   - Challenges:
     - Often not a well-formed sentence; code snippets as nouns/verbs
     - Referents are implicit, assumes coding knowledge
2. Match each subject to a Java element
   - Pattern matching
   - Lexical similarity to identifiers, types, documentation
3. Match each predicate to a Java element
4. Create assert statement from expressions and methods

Results

On 381 **`@throws`** clauses:

- 82% precision
- 57% recall

Can tune parameters to favor either metric

Pattern-matching and pre-processing are important

Current work:

- **`@param`** and **`@return`** tags
- Integrate with Randoop test generator

Related work

Heuristics

- JCrasher, Crash’n’Check (Csallner, and Smaragdakis. ICSE ’05)
- Randoop (Pacheco, Lahiri, Ernst, and Ball. ICSE ’07)

Specifications

- ASTOOT (Doong, and Frankl. TOSEM ’94)
- Models, contracts, …

Properties

- Cross-checking oracles (Carzaniga, Goffi, Gorla, Mattavelli, and Pezzè. ICSE ’14)
- Metamorphic testing (Chen, Kuo, Tse, and Zhou. STEP ’13)
- Symmetric testing (Gotlieb. ISSRE ’03)

Natural language documentation

- @tComment (Tan, Marinov, Tan, and Leavens. ICST ’12)
- aComment (Tan, Zhou, and Padioleau. ICSE ’11)
- iComment (Tan, Yuan, Krishna, and Zhou. SOSP ’07)

Applying NLP to software engineering

Problems	NL sources	NLP techniques
inadequate diagnostics	error messages	document similarity
incorrect operations	variable names	word semantics
missing tests	code comments	parse trees
unimplemented functionality	user questions	translation

Machine translation

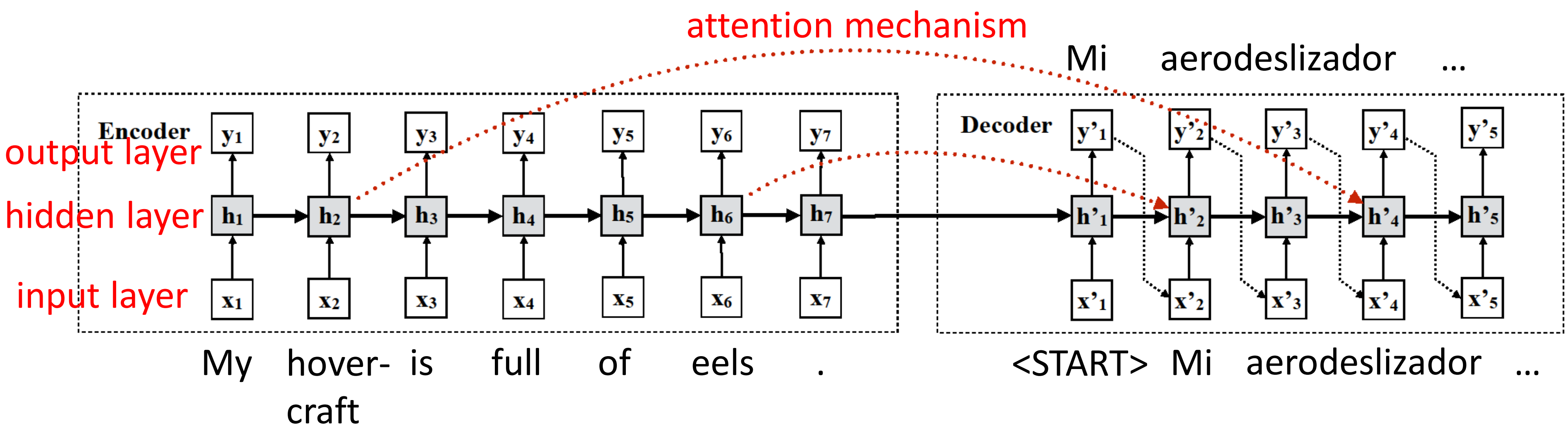
English: “My hovercraft is full of eels.”

Spanish: “Mi aerodeslizador está lleno de anguilas.”

English: “Don’t worry.”

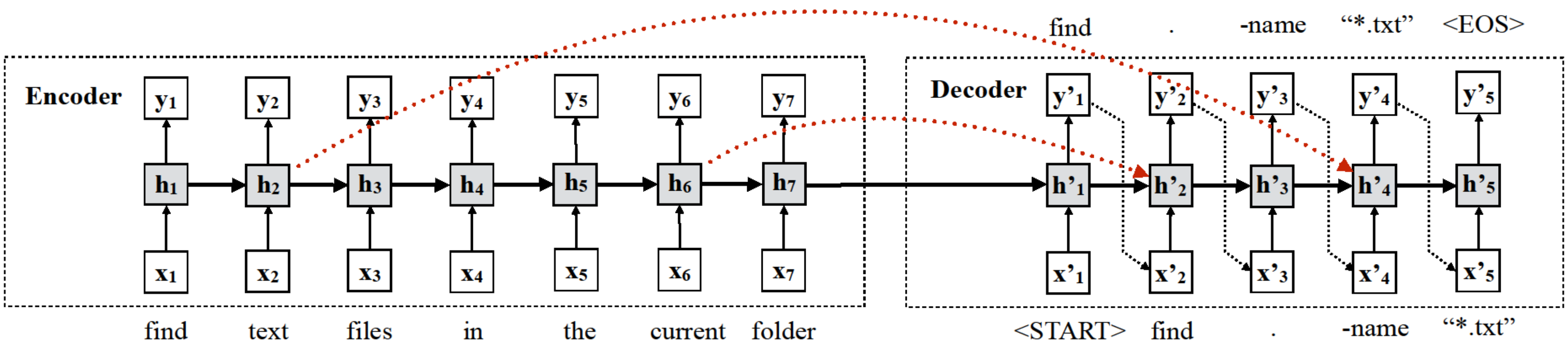
Spanish: “No te preocupes.”

Sequence-to-sequence recurrent neural network translators



Input, hidden, and output functions are inferred from training data using probability maximization.

Tellina: text to commands



- Training data: ~8000 ⟨text, command⟩ pairs
  - Collected manually from webpages, plus cleaning
- Uses of **`find`** and English descriptions
  - Compound commands: **`()`**, **`&&`**, **`||`**
  - Nesting: **`|`**, **`$()`**, **`<()`**
  - Strings are opaque; no command interpreters (**`awk`**, **`sed`**)
  - No bash compound statements (**`for`**)

Results

Accuracy for Tellina’s first output:

- Structure of command (without constants): 69%
- Full command (with constants): 30%

User experiment:

- Tellina makes users 22% more efficient
  - Even though it lies 1/3 of the time
- Qualitative feedback
  - Most participants wanted to continue using Tellina (5.8/7 Likert scale)
  - Partially-correct answers were helpful, not too hard to correct
  - Output bash commands are sometimes not syntactic or subtly wrong
  - Needs explanation of meaning of output bash commands

Related work

Neural machine translation

- Sequence-to-sequence learning with neural nets [Sutskever 2014]
- Attention mechanism [Luong 2015]

Semantic parsing

- Translating natural language to a formal representation [Zettlemoyer 2007, Pasupat 2016]

Translating natural language to DSLs

- If-this-then-that recipes [Quirk 2015]
- Regular expressions [Locascio 2016]
- Text editing, flight queries [Desai 2016]

Other software engineering projects

- Analyzing programs before they are written
- Gamification (crowd-sourcing) of verification
- Evaluating and improving fault localization
- Pluggable type-checking for error prevention

- … many more: systems, synthesis, verification, etc.

UW is hiring! Faculty, postdocs, grad students

Applying NLP to software engineering

Problems	NL sources	NLP techniques
inadequate diagnostics	error messages	document similarity
incorrect operations	variable names	word semantics
missing tests	code comments	parse trees
unimplemented functionality	user questions	translation

Machine learning + software engineering

- Software is more than source code
- Formal program analysis is useful, but insufficient
- Analyze and generate all software artifacts

A rich space for further exploration