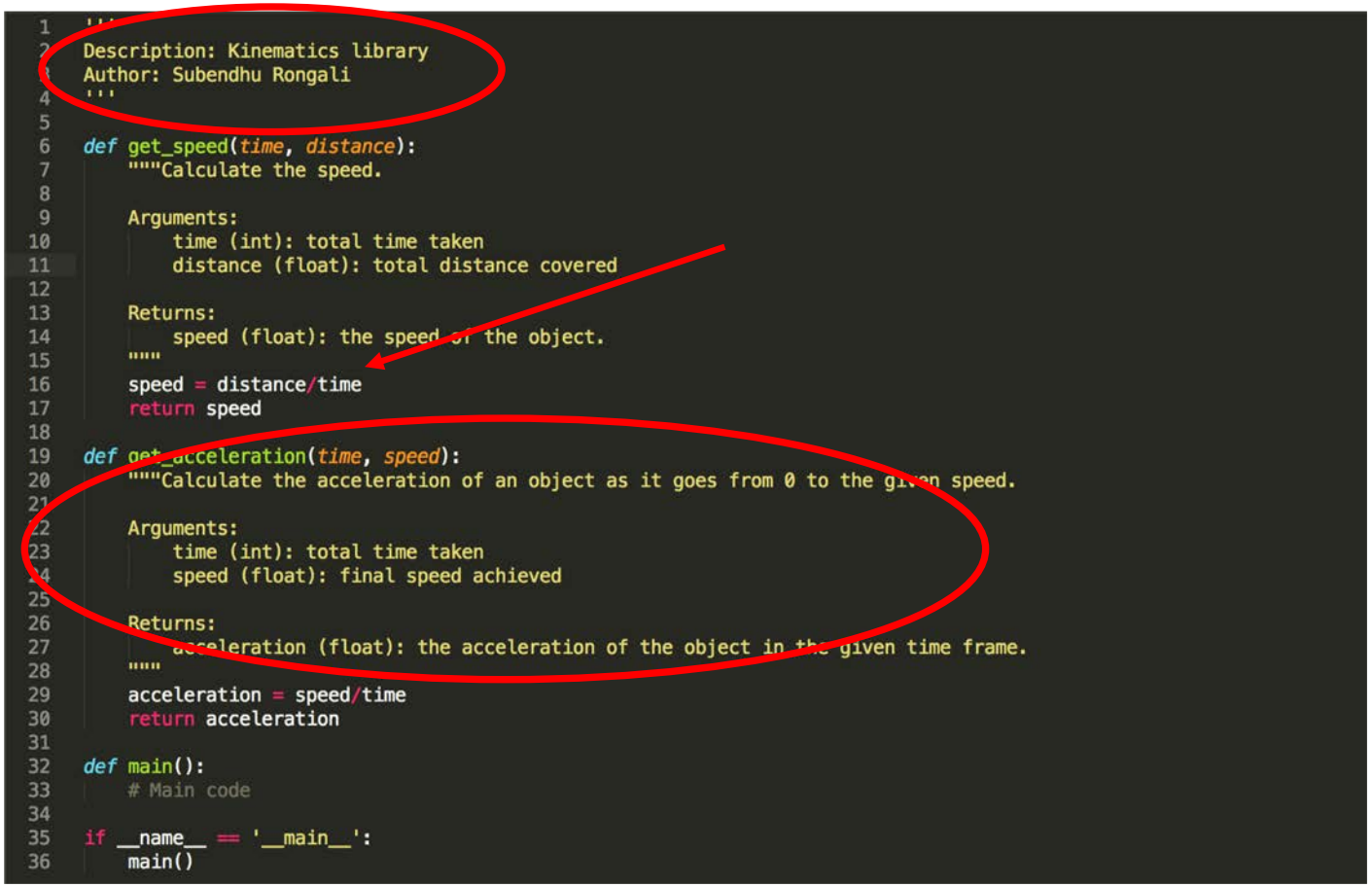
Natural language is a programming language: Applying natural language processing to software development

Michael D. Ernst

Presented by: Tomas Geffner, Subendhu Rongali & Natcha Simsiri

Before we start, what is software?



```
'''
Description: Kinematics library
Author: Subendhu Rongali
'''

def get_speed(time, distance):
    """Calculate the speed.

    Arguments:
        time (int): total time taken
        distance (float): total distance covered

    Returns:
        speed (float): the speed of the object.
    """
    speed = distance/time
    return speed

def get_acceleration(time, speed):
    """Calculate the acceleration of an object as it goes from 0 to the given speed.

    Arguments:
        time (int): total time taken
        speed (float): final speed achieved

    Returns:
        acceleration (float): the acceleration of the object in the given time frame.
    """
    acceleration = speed/time
    return acceleration

def main():
    # Main code

if __name__ == '__main__':
    main()
```

Not just code/AST

It is also:

- test cases
- documentation
- variable names
- program structure
- the version control repo
- the issue tracker
- conversations
- user studies
- program executions

..and much more

Issue

How do we create better software tools?

Previous tools mostly depend on the ASTs of code.

But software isn’t just ASTs of the code!

Why not look at software more comprehensively?

Research Problem

Can we create better software development tools using additional artifacts developers create (e.g documentation, bug report, etc)?

Some common problems - inadequate error messages, incomplete test suite etc.

Key Idea & Contributions

Use NLP techniques to analyse the natural language embedded in the software and solve some problems.

NLP based solutions for four common software problems.

- Detection of inadequate diagnostic messages
- Identifying undesired variable interactions
- Generation of test oracles
- Generating code from natural language specifications

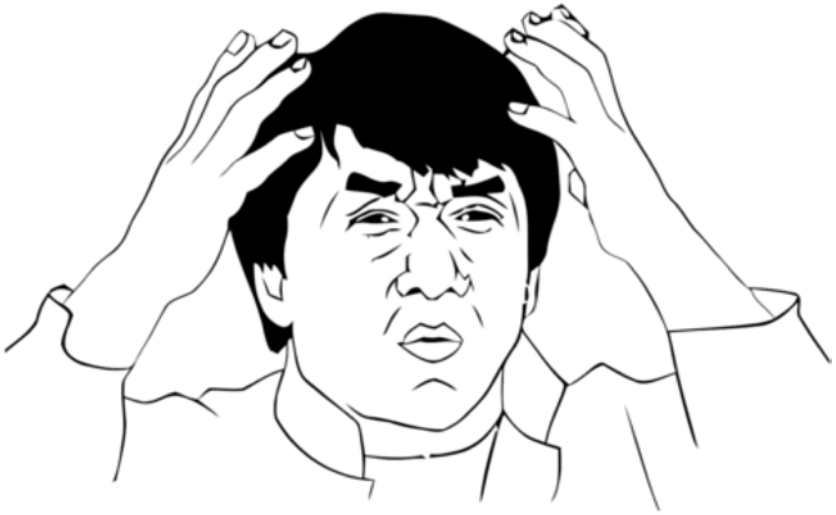
Detection of inadequate diagnostic messages

$ python route.py -port_num=100

~~unexpected system failure~~

~~your port number sux lol~~

your port number is already in use



Inadequate diagnostic messages waste 25% of a software maintainer's time!

What can we do?

ConfDiagDetector: Tells you if your error messages are adequate.

Configuration mutation + NLP

Mutate a configuration option to get an error

Doc similarity between configuration option description and the error message

Evaluation - does it work?

ConfDiagDetector reported **25 missing and 18 inadequate messages** in four open-source projects: Weka, JMeter, Jetty, and Derby.

Validation by three programmers indicated a **0% false negative rate and a 2% false positive rate** (previous best tool has 16% false positive).

Previous methods all troubleshoot an exhibited error or require lots of help like source code, usage history and OS-level support.

Identifying Undesired Variable Interactions

Incompatible variable interactions are a common mistake.

ex: totalPrice = itemPrice + shippingDistance

You can tell it’s wrong looking at the variable names.

What can we do?

Ayudante: Clusters the variables in two ways.

1) ***NLP based*** - Tokenize words, compute similarity using WordNet or edit distance

2) ***Abstract Type Inference*** - Variables that interact with each other in code (ex. x < y)

Identify discrepancies between clusters

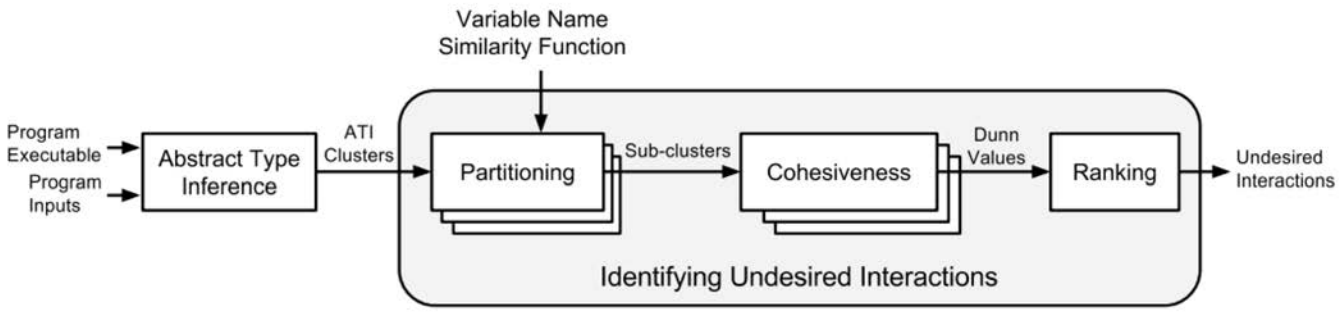


Figure 1 Ayudante architecture.

Evaluation - does it work?

Ayudante’s top-ranked report about the grep program indicated an interaction in grep that was likely undesired, because it discards information.

There are variable naming conventions. Some languages allow storing units.

No exact previous work. Components like tokenization outperform prior methods.

Generation of test oracles

Programmers don’t like writing not-code.

Manual test suites neglect important behavior. Automatic ones lack gold standard.

What can we do?

Let’s use code comments - Javadoc comments, templates

ToraDocu: Convert sentences into assertions - English to code using parse trees!

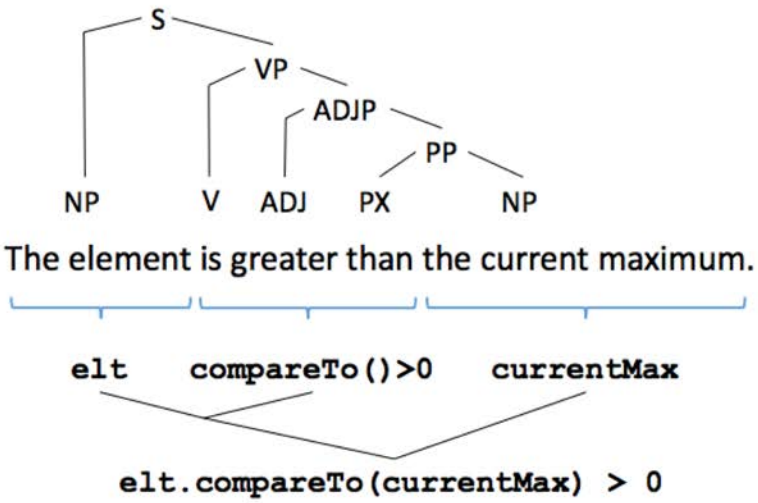


Figure 2 Parsing a sentence and unparsing into an assertion.

Evaluation - does it work?

941 programmer-written Javadoc specifications - **88% precision and 59% recall** in translating them to executable assertions

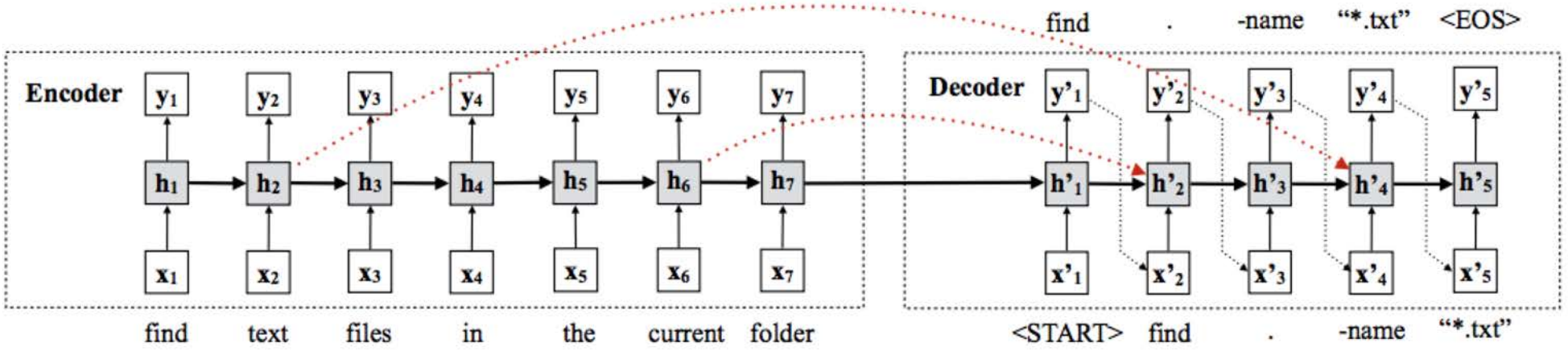
Improved the fault-finding effectiveness of EvoSuite and Randoop test suites by **8% and 16%**

Sophisticated NLP - better than simple pattern matching techniques.

Generating code from natural language specifications

What can we do?

Tellina: Neural machine translation from english to code using RNNs.



Evaluation - does it work?

Convert english specifications of file systems operations to bash.

Trained on 5000 <text, bash> pairs from StackOverflow and bash tutorials.

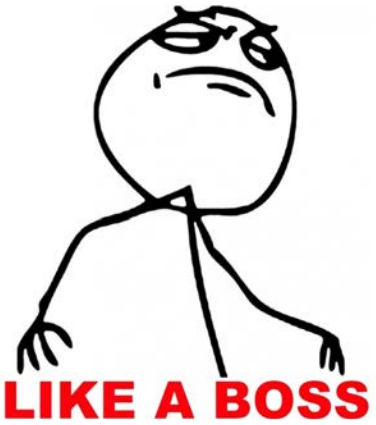
Top-1 and top-3 accuracy for the structure of the command - **69% and 80%**

Some errors - but still useful to programmers!

Previous works were on simple languages, regexes etc.

Summary

Catchy names for tools



		Problem	NL source	NLP technique
§3	Analyze existing	inadequate diagnostics	error messages	document similarity
§4	code to find bugs	incorrect operations	variable names	word semantics
§5	Generate	missing tests	code comments	parse trees
§6	new code	unimplemented functionality	user questions	translation

Discussion questions

1. Can we do direct translation for code that's more than a single line?

Discussion questions

2. Do we really create test oracles if we only have 88% precision?

Discussion questions

3. Can we trust programmers to use good variable names? Can we improve their method?

Discussion questions

4. Can we use translation instead of parse trees for problem 3?

Discussion questions

5. We only analyze diagnostic messages for configuration option erros. Can we do this for any error in general?