AI in Software Engineering

Prof. Dr. Holger Giese and Christian Medeiros Adriano
Head of the System Analysis and Modeling Group
Hasso Plattner Institute at the University of Potsdam
Software Engineering for AI vs. AI in Software Engineering

models/natural language

Generate tests

Generate code

code

compile

executable

tests

compile

executable

code

analyze

outcomes

AI in Software Engineering
Prof. Holger Giese
Hasso Plattner Institute
Chart 2


Self-Adaptive & Train Goals

1) Training
2) Evaluation (meta level)

Utility Change Predictor

predictions impact of adaptation rules

Adaptation Engine

observes Simulation

Managed resource

predicts impact of adaptation rules

R Studio simulates

Managed resource

Provides ground truth

0
0
0
0

Linear Saturating Discontinuous Combined

Managed resource

Control A

Simulator

Adaptation Engine

R Studio

Configuration control

Managed resource

Utility Change Predictor

Simulator generates prediction models (.pmml )

Managed resource generates data for machine learning

Adaptation Engine observes

Control B

Control C

Adaptation Engine observes

Managed resource adapts

Managed resource adapts

Managed resource


Software Engineering for AI
We worked on...
How to optimally allocate code inspection task to minimize cost and time while maximizing the accuracy of software failure diagnostic?

**Approach:** causal and sequential decision models decide which tasks to generate and who should execute them.
Software Engineering for AI vs. AI in Software Engineering

- Code
- Generate code
- Generate tests
- Models/natural language
- Analyze
- Outcomes
- Tests
- Executable
- Compile

AI in Software Engineering
Prof. Holger Giese
Hasso Plattner Institute
Chart 5
Agenda

1. Background: Automation in SE
2. Overview: AI in SE
3. Use Case 1: Clone Detection
4. Use Case 2: Code Completion
5. Use Case 3: Code Generation
6. Conclusion & Outlook
1. Background: Automation in SE

Disruptive automations in software engineering industry

**Observations:**

- **Automation** that affect the executable is more “dangerous” (generate code, compile, …)

- **Automation** that do not affect the executable is less “dangerous” as it may only affect the executable via human decisions (generate tests, …)
2. Overview: AI in SE

Fig. 2. The distribution of DL techniques in Different SE activities.

Fig. 3. The classification of primary studies.

<table>
<thead>
<tr>
<th>Task type</th>
<th>Data type</th>
<th>Dataset</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>defect prediction</td>
<td>code-based data</td>
<td>PROMISE dataset</td>
<td>[SP04, IEEE46, IEEE43, IEEE85, IEEE127]</td>
</tr>
<tr>
<td></td>
<td>code-based data</td>
<td>Ivy, jEdit, Log4j, Lucene, PBears, POI, Synapse, Velocity, Xalan-J, Xerces</td>
<td>[IEEE87, IEEE92]</td>
</tr>
<tr>
<td></td>
<td>code-based data</td>
<td>cleaned NASA, AEEEM datasets</td>
<td>[SP04]</td>
</tr>
<tr>
<td></td>
<td>code-based data</td>
<td>Bugzilla, Columbus, Eclipse JDT, Eclipse Platform, Mozilla and PostgreSQL</td>
<td>[IEEE44]</td>
</tr>
<tr>
<td></td>
<td>text-based data</td>
<td>CSIC dataset</td>
<td>[IEEE10]</td>
</tr>
<tr>
<td>program repair</td>
<td>code-based data</td>
<td>DeepFix dataset</td>
<td>[AAA193, IEEE118]</td>
</tr>
<tr>
<td></td>
<td>code-based data</td>
<td>SATE IV</td>
<td>[MIT01]</td>
</tr>
<tr>
<td></td>
<td>code-based data</td>
<td>ETH-Py1502, MSR-VarMissuse</td>
<td>[ICLR02]</td>
</tr>
<tr>
<td></td>
<td>code-based data</td>
<td>SPoC dataset</td>
<td>[ACM11]</td>
</tr>
<tr>
<td></td>
<td>code- and text-based data</td>
<td>Bugs, QuickBugs, ManySSimBugsJ</td>
<td>[IEEE89, IEEE46]</td>
</tr>
<tr>
<td></td>
<td>code- and text-based data</td>
<td>Bugs.jac</td>
<td>[IEEE29, IEEE46]</td>
</tr>
<tr>
<td></td>
<td>code-based data</td>
<td>OJClone</td>
<td>[AAA59, MK04, MK05, ACM28, IEEE38]</td>
</tr>
<tr>
<td></td>
<td>code-based data</td>
<td>Google Code Jam</td>
<td>[ACM64, IEEE22]</td>
</tr>
<tr>
<td>bug report related</td>
<td>text-based data</td>
<td>Summary Dataset(SDS)</td>
<td>[IEEE137, IEEE59]</td>
</tr>
<tr>
<td></td>
<td>text-based data</td>
<td>Authorship Dataset (ADS)</td>
<td>[IEEE137, IEEE59]</td>
</tr>
<tr>
<td></td>
<td>text-based data</td>
<td>OpenOffice, Eclipse, Net Beans</td>
<td>[IEEE139]</td>
</tr>
<tr>
<td>self-admitted technical debt</td>
<td>text-based data</td>
<td>[14]</td>
<td>[IEEE100]</td>
</tr>
<tr>
<td>detection</td>
<td>text-based data</td>
<td>[1]</td>
<td>[IEEE28]</td>
</tr>
<tr>
<td>code review</td>
<td>code-based data</td>
<td>cloudstack, ambari,aurora, drillgit, accumulo and libbase-git.</td>
<td>[AAA195]</td>
</tr>
<tr>
<td>code change</td>
<td>text-based data</td>
<td>QT, OPENSTACK datasets</td>
<td>[IEEE45]</td>
</tr>
<tr>
<td>software/code classification</td>
<td>code-based data</td>
<td>[AAA102]</td>
<td>[AAA192]</td>
</tr>
</tbody>
</table>
### AI in SE: Software Development

<table>
<thead>
<tr>
<th>Task type</th>
<th>Data type</th>
<th>common dataset</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code representation</td>
<td>code-based data</td>
<td>Java-med</td>
<td>[ICLR07]</td>
</tr>
<tr>
<td></td>
<td>code-based data</td>
<td>eth_py150_open</td>
<td>[ACM18, ACM38]</td>
</tr>
<tr>
<td></td>
<td>code-based data</td>
<td>Code.org’s Hour of Code (HOC)</td>
<td>[ACM20]</td>
</tr>
<tr>
<td>Code generation</td>
<td>code-based data</td>
<td>HearthStone (HS)</td>
<td>[SP09, MITP06]</td>
</tr>
<tr>
<td></td>
<td>code-based data</td>
<td>Karel dataset</td>
<td>[ICLR09, MITP08, ICLR03]</td>
</tr>
<tr>
<td></td>
<td>code-based data</td>
<td>Spider</td>
<td>[MITP06]</td>
</tr>
<tr>
<td></td>
<td>code-based data</td>
<td>DeepCom</td>
<td>[MITP07]</td>
</tr>
<tr>
<td>Code comment generation</td>
<td>code-based data</td>
<td>Google Code Jam</td>
<td>[AAAI08]</td>
</tr>
<tr>
<td></td>
<td>text-based data</td>
<td>WMT19</td>
<td>[IEEE36]</td>
</tr>
<tr>
<td></td>
<td>code- and text-based data</td>
<td>CODEnn</td>
<td>[IEEE36]</td>
</tr>
<tr>
<td>Code search</td>
<td>code- and text-based data</td>
<td>StaQC benchmark</td>
<td>[IEEE02]</td>
</tr>
<tr>
<td></td>
<td>code- and text-based data</td>
<td>CODEnn</td>
<td>[IEEE24]</td>
</tr>
<tr>
<td></td>
<td>code- and text-based data</td>
<td>CoSbench</td>
<td>[IEEE24]</td>
</tr>
<tr>
<td>Code completion</td>
<td>code-based data</td>
<td>JavaScript (JS)</td>
<td>[MK09]</td>
</tr>
<tr>
<td></td>
<td>code-based data</td>
<td>Python (PY)</td>
<td>[MK09]</td>
</tr>
<tr>
<td>Code localization</td>
<td>image-based data</td>
<td>YouTube</td>
<td>[SP08, ACM24, ACM36, IEEE140]</td>
</tr>
<tr>
<td>Code summarization</td>
<td>code-based data</td>
<td>NCF representation</td>
<td>[ICLR06]</td>
</tr>
<tr>
<td></td>
<td>code- and text-based data</td>
<td>LeClair et al.</td>
<td>[IEEE136]</td>
</tr>
<tr>
<td>Method name generation</td>
<td>code- and text-based data</td>
<td>MCC corpus</td>
<td>[IEEE59]</td>
</tr>
</tbody>
</table>

3. Use Case 1: Clone Detection

1. **Type-1** (Textual similarity) = Identical source code fragments (ignore white-space, layout and comments)

2. **Type-2** (Lexical, token-based, similarity) = Identical source code fragments (ignore differences in identifier names)

3. **Type-3** (Syntactic similarity) = Source code fragments that differ at the statement level, e.g., fragments can have statements added, modified and/or removed.

4. **Type-4** (Semantic similarity) = Syntactically distinct source code fragments, **but** that implement the same functionality


**Benchmark BigCloneBench**

![Chart 13](image)

**Easy to Detect**

**Difficult to Detect**

<table>
<thead>
<tr>
<th>Clone Type</th>
<th>Number</th>
<th>Percent(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>48062</td>
<td>0.56</td>
</tr>
<tr>
<td>T2</td>
<td>4614</td>
<td>0.054</td>
</tr>
<tr>
<td>VST3</td>
<td>4182</td>
<td>0.053</td>
</tr>
<tr>
<td>ST3</td>
<td>16775</td>
<td>0.193</td>
</tr>
<tr>
<td>MT3</td>
<td>86109</td>
<td>1.0</td>
</tr>
<tr>
<td><strong>WT3/T4</strong></td>
<td><strong>8416032</strong></td>
<td><strong>98.14</strong></td>
</tr>
</tbody>
</table>

A software S is represented as a collection of code blocks $S : \{B_1, ..., B_n\}$, where each $B_i$ corresponds to a bag-of-tokens $B : \{T_1, ..., T_k\}$, where $T$ is token (method, variable, operator names, etc.)

**Assumption:** source code follows the Zipf law (similarly as natural language), which preconizes that there are few very popular tokens, and the frequency of tokens decreases very rapidly with popularity rank (Figure-1)

**Insight:**
- Most code blocks contain one or more of the few very popular tokens (e.g., keywords, counters like i, j)
- Few code blocks share rare tokens (e.g., identifiers that are domain or project specific).
- Hence, if we sort code blocks by the popularity of tokens in the corpus, the sub-blocks will consist of these rare tokens. This will ensure low probability of different sub-blocks having a similar token.

---

**Figure-1 power-law like distribution of token frequency (popularity)**


http://wugology.com/zipfs-law/
Hybrid Rule and Learning-based – Oreo

- **Size Similarity**: number of tokens
- **Semantic Similarity**: number of actions tokens (function signature – `getByte()`, `toString()`, etc.) shared by two methods
- **Metrics Similarity**: Halstead effort, Halsted difficulty, Cyclomatic complexity, etc.

![Diagram showing the process of clone detection using Size, Semantic, and Metrics similarity](chart)

- **Candidates**
  - Pre-Processing
  - Metric Filter Match
  - Deep Neural Network

- **Type 1 & 2**
- **Type 3 & 4**

**AI in Software Engineering**
Prof. Holger Giese
Hasso Plattner Institute

Saini, V., et al., 2018, Oreo: Detection of clones in the twilight zone, Proceedings of the 26th ACM ESC/FCE
Examples with Oreo

Listing 3: Clone Pair Example: 1

```java
private void sortByName() {
    int i, j;
    String v;
    for (i = 0; i < count; i++) {
        ChannelItem ch = chan[i];
        v = ch.getTag();
        j = i;
        while ((j > 0) & (collator.compare(chan[j - 1].getTag(), v) > 0)) {
            chan[j] = chan[j - 1];
            j--;
        }
        chan[j] = ch;
    }
}

public void bubblesort(String filenames[]) {
    for (int i = filenames.length - 1; i > 0; i--)
        for (int j = 0; j < i; j++)
            if (filenames[j].compareTo(filenames[j + 1]) > 0) {
                String temp;
                temp = filenames[j];
                filenames[j] = filenames[j + 1];
                filenames[j + 1] = temp;
            }
}
```

Listing 4: Clone Pair Example: 2

```java
public static String getExtension(String filename) {
    if (filename == null || filename.trim().length() == 0 || !filename.contains(".")) return null;
    int pos = filename.lastIndexOf(".");
    return filename.substring(pos + 1);
}

private static String getFormatByName(String name) {
    if (name != null) {
        final int j = name.lastIndexOf(".") + 1, k = name.lastIndexOf("/") + 1;
        if (j > k && j < name.length()) return name.substring(j);
    }
    return null;
}
```

Listing 5: False Positive Example

```java
public static String getHexString(byte[] bytes) {
    if (bytes == null) return null;
    StringBuilder hex = new StringBuilder(2 * bytes.length);
    for (byte b : bytes) {
        hex.append(HEX_CHARS[(b & 0xf0) >> 4]).append(HEX_CHARS[(b & 0xf0)]);
    }
    return hex.toString();
}
```

Saini, V., et al., 2018, Oreo: Detection of clones in the twilight zone, Proceedings of the 26th ACM ESC/FCE
Only Learning-based – Sia-RAE
Clone Detection: Discussion

Clone Detection Capabilities

- Rule-based approaches can detect simpler types (T1, T2)
- Learning-based approaches can detect more complex types (T3, T4)
- Deep Learning permits to better detect in particular hard types (ST3, MT3, WT3, WT4)

Handling False Positives

- The results must be still manually checked, which can become quite challenging in case of complex/hard cases

Studies about the productivity gains due to better clone detection with AI are missing; likely as the long-term impact on maintenance is very hard to evaluate ...
1. **Goal**: given a code context, predict the next token, next line.

2. **Token**: names of methods, variables, operations, types

3. **Facts affecting training** [Allamanis & Sutton 2013]
   - Distribution - Most frequent tokens are few, whereas low frequency are plentiful
   - Power-law - The frequency is a logarithm function of the count of token appearances
   - Context - The more frequent tokens are more context dependent

---

**Use Case 2: Code Completion**

1. More frequent terms are easier to predict given a context.
2. Code Completion is less certain when predicting domain-specific variable names, because they are less frequent.

Probability assigned to next token

Easy

1  public void execute(Runnable task) {
2   if (task == null)
3     throw new NullPointerException();
4   ForkJoinTask<?> job;
5   if (task instanceof ForkJoinTask<?>) // avoid re-wrap
6       job = (ForkJoinTask<?>) task;
7     else
8       job = new ForkJoinTask<AdaptedRunnableAction(task);
9       doSubmit(job);
10  }

Code Completion: Traditional vs Deep-Learning

Traditional/Statistical (Built in most IDES)

- **Input**: Previous token
- **Output**: Next token or Next line

Deep Learning-Based (CoPilot, Tabnine, Kite)

- **Input**: Entire source code and Text comments
- **Output**: Naming, code blocks and entire methods

Text-based

Type-based

Colored – Typed by user
Grey - Recommendation


Code Completion: Deep Learning-Based – CoPilot (1/3)

Colored – Typed by user
Grey - Recommendation

Suggests a whole set of parameters for the function with an explanation.

Suggests the entire body of the function

GitHub - https://docs.github.com/en/copilot
6 ways GitHub Copilot helps you write better code faster - https://www.youtube.com/watch?v=SxtMnn1v7d8
**Code Completion: Deep Learning-Based – CoPilot (2/3)**

**Step 1:** Existing Test Class

```javascript
import './model.js';
import { deepEqual } from 'assert';

it('should return an empty array by default', () => {
    const model = new Model();
    deepEqual(model.get(), []);
});

it('should add double the number to the list', () => {
    const model = new Model();
    model.add(7);
    deepEqual(model.get(), [14]);
});
```

**Step 2:** Suggests a whole class based on the file (model.test.js) that existed.

```javascript
export default class Model {
    constructor() {
        this.list = [];
    }

    get() {
        return this.list;
    }

    add(number) {
        this.list.push(number * 2);
    }
}
```

!!! Knows that the class should add double the number (number * 2)

[GitHub](https://docs.github.com/en/copilot)

6 ways GitHub Copilot helps you write better code faster - [https://www.youtube.com/watch?v=SXtMnn1v7d8](https://www.youtube.com/watch?v=SXtMnn1v7d8)
What do programmers think about it? New Ways of Programming?

“LLM-assisted programming shares some properties of compilation, pair programming, and programming via search and reuse, there are fundamental differences both in the technical possibilities as well as the practical experience.”.


Faster: “I think of Copilot as an intelligent autocomplete... I already have the line of code in mind and I just want to see if it can do it, type it out faster than I can.”

Reuse: “Copilot feels useful for doing novel tasks that I don’t necessarily know how to do. It is easier to jump in and get started with the task”.

Nonetheless... “I was about to write the code and I knew what I wanted to write. But now I’m sitting here, seeing if somehow Copilot came up with something better than the person who’s been writing Haskell for five years, I don’t know why am I giving it the time of day.”

Code Completion - Discussion

Productivity with Code Completion

- Recruited software developers were asked to implement an HTTP server in JavaScript as quickly as possible. The treatment group, with access to the AI pair programmer, completed the task 55.8% faster than the control group.


Quality of Code Completion

- Copilot-generated code is harder to debug
- Programmers validate suggestions by “pattern matching”
- Programmers are reluctant to accept or repair suggestions.

Risks of Code Completion

- Programmers suffer from an anchoring bias when looking through multiple suggestions.
- Programmers suffer from cognitive overload due to multi-suggestion pane.

5. Use Case 3: Code Generation

ChatGPT = InstructGPT + RLHF

1. ChatGPT = conversational AI, which is very challenging because human language is ambiguous, and conversations are bounded by contextual information.

2. Instruct GPT = language model trained to follow instructions from a prompt

3. RLHF (Reinforcement Learning from Human Feedback) = human gives feedback on a generated text in order to align the pre-trained language model with complex human values (what is funny, ethical, safe).

Step 1: Pre-Train a Language

Step 2: Train a Reward Model

Step 3: Fine-Tune with RL

Very large investment on crowdsourcing annotation work

Lambert et al., 2022, Illustrating Reinforcement Learning from Human Feedback (RLHF), Hugging Face- https://github.com/huggingface/blog/blob/main/rlhf.md


AI in Software Engineering
Prof. Holger Giese
Hasso Plattner Insitute
Chart 26
Write a program that "shuts off the pumps if the water level remains above 100 m for more than 4 seconds."

1st version

As an AI language model, I don't have direct access to the physical pumps and water level sensors to implement the program. However, I can provide a basic pseudocode implementation that describes the logic of the program:

```java
set pump_status = "off"
set water_level_timer = 0

while true:
    water_level = read_water_level_sensor()
    if water_level > 100:
        water_level_timer += 1
    else:
        water_level_timer = 0

    if water_level_timer >= 4:
        pump_status = "off"
        turn_off_pumps()
    else:
        pump_status = "on"
        turn_on_pumps()

    wait_for_next_iteration()
```

Write a program that “shuts off the pumps if the water level remains above 100 m for more than 4 seconds.”

2nd version

I’m sorry, as an AI language model, I do not have the capability to directly control physical devices or write executable programs that interact with the physical world. However, I can provide a high-level algorithm that could be used to achieve the desired functionality:

1. Continuously monitor the water level sensor.
2. If the water level is above 100 m, start a timer and set a flag indicating that the water level has been above 100 m.
3. If the water level drops below 100 m, reset the timer and clear the flag.
4. If the timer reaches 4 seconds and the flag is still set, shut off the pumps.

This algorithm can be implemented using a variety of programming languages, platforms, and hardware components, depending on the specific requirements of the application.

Write a program that "shuts off the pumps if the water level remains above 100 m for more than 4 seconds."

3rd version

As an AI language model, I don't have direct access to the physical pumps and water level sensors to implement the program. However, I can provide a basic pseudocode implementation that describes the logic of the program:

```python
import time

# Initialize water level and pump status
water_level = 0
pumps_on = False
pump_on_time = 0

# Loop to simulate water level changes
while True:
    # Check water level
    if water_level > 100:
        # Turn pumps off if they are on
        if pumps_on:
            pumps_on = False
            print("Pumps turned off")
        # Record time if pumps are off
        else:
            pump_off_time = time.time()
    # Check if water level has been above 100 for more than 4 seconds
    if pump_off_time - pump_on_time > 4:
        print("Water level above 100 for more than 4 seconds. Pumps turn back")
    else:
        # Turn pumps on if they are off
        if not pumps_on:
            pumps_on = True
            pump_on_time = time.time()
            print("Pumps turned on")

    # Simulate water level changes
    water_level += 1
    time.sleep(1)
```

**Code Generation (4/6)**

```
java
set pump_status = "off"
set water_level_timer = 0

while true:
    water_level = read_water_level_sensor()
    if water_level > 100:
        water_level_timer += 1
    else:
        water_level_timer = 0
    if water_level_timer >= 4:
        pump_status = "off"
        turn_off_pumps()
    else:
        pump_status = "on"
        turn_on_pumps()

wait_for_next_iteration()
```

1st version

**Python**

```python
import time

# Initialize water level and pump status
water_level = 0
pumps_on = False
pump_on_time = 0

# Loop to simulate water level changes
while True:
    # Check water level
    if water_level > 100:
        # Turn pumps off if they are on
        if pumps_on:
            pumps_on = False
            print("Pumps turned off")
        # Record time if pumps are off
        else:
            pump_off_time = time.time()
    # Check if water level has been above 100 for more than 4 seconds
    if pump_off_time - pump_on_time > 4:
        print("Water level above 100 for more than 4 seconds. Pumps turned back")
    else:
        # Turn pumps on if they are off
        if not pumps_on:
            pumps_on = True
            pump_on_time = time.time()
            print("Pumps turned on")
    # Simulate water level changes
    water_level += 1
    time.sleep(1)
```

3rd version

---

**AI in Software Engineering**

Prof. Holger Giese
Hasso Plattner Institute

30

Write a program in Java to shut off the pumps if the water level remains above 100 m for more than 4 seconds.

Sure, here's an example Java program that can shut off the pumps if the water level remains above 100 m for more than 4 seconds:

```java
import java.util.Timer;
import java.util.TimerTask;

public class WaterLevelMonitor {
    private static final int PUMP_PIN = 12;
    private static final int WATER_LEVEL_PIN = 18;
    private static final int WATER_LEVEL_THRESHOLD = 100;
    private static final int SHUTOFF_DELAY = 4000; // in milliseconds

    private Timer shutoffTimer;

    public WaterLevelMonitor() {
        // Set up GPIO pins here if necessary
    }

    public void run() {
        // Check water level
        boolean waterLevelAboveThreshold = checkWaterLevel();
        if (waterLevelAboveThreshold) {
            // Water level has been above threshold for more than SHUTOFF_DELAY
            shutoffPump();
        } else if (waterLevelAboveThreshold) {
            // Water level has been below threshold
            shutoffPump();
        } else if (waterLevelAboveThreshold) {
            // Water level has been above threshold
            shutoffPump();
        }
        shutoffTimer.schedule(new TimeTask(), 0);
    }

    private boolean checkWaterLevel() {
        // Implement code to read water level from sensor
        return true; // Return true if water level is above threshold, false otherwise
    }

    private void shutoffPump() {
        // Implement code to shut off pump
    }

    public static void main(String[] args) {
        WaterLevelMonitor monitor = new WaterLevelMonitor();
        monitor.startMonitoring();
        // Keep process running until interrupted
        while (true) {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                break;
            }
        }
        monitor.stopMonitoring();
    }
}
```

Note that this code assumes you have a way to read the water level sensor input and control the pump. You may need to modify the code to match your specific hardware and libraries. Additionally, please test and validate any code you use for controlling important equipment like pumps, and follow appropriate safety protocols.

“Shut off the pumps if the mean water level over the past 4 seconds was above 100 m.”

\[ \frac{\int_{\tau-4}^{\tau} WL(t) dt}{4} > 100 \]

“Shut off the pumps if the median water level over the past 4 seconds was above 100 m.”

\[ \frac{\text{MAX}_{(\tau-4, \tau)}[WL(t)] + \text{MIN}_{(\tau-4, \tau)}[WL(t)]}{2} > 100 \]

“Shut off the pumps if the minimum water level over the past 4 seconds was above 100 m.”

\[ \text{MIN}_{(\tau-4, \tau)}[WL(t)] > 100 \]

Ambiguity and natural language ➔ Many possible interpretations ...

Code Generation - Discussion

- Code has still to be adjusted
  - Positioned in the code
  - Replace variables
  - Use the right functions/operations
  - ...
- Code may be wrong
  - Pumps are shut off too early in version 2
- Best code offered (1st, 2nd, ...) must be identified and adapted/corrected
- **Reuse?** But produces many clones...

```java
set pump_status = "off"
set water_level_timer = 0
while true:
    water_level = read_water_level_sensor()
```

```python
import time
# Initialize water level and pump status
water_level = 0
pumps_on = False
pump_on_time = 0

# loop to simulate water level changes
while True:
    # check water level
    if water_level > 100:
        # turn pumps off if they are on
        if pumps_on:
            pumps_on = False
            print("Pumps turned off")
        # record time if pumps are off
```
6. Conclusion & Outlook

- Automation will substantially improve using AI (resp. has already)
- **Automation** that **does not affect** the executable can therefore **better** support human decisions
  - **Pro:** better clone detection can result in **better** maintenance decisions
  - **Cons:** too much **trust** can also here lead also to wrong decisions
- **Automation** that **affect** the executable raises very subtle problems
  - humans must **evaluate** the outcomes for the specific context (e.g., check that the code really does what is needed (do they know? corner-cases?),
  - humans must **adapt** the outcomes to the specific context (e.g., replace variables (may become quite complex)), and
  - humans may have to **change** the outcomes later or regenerate place them (**maintenance** may become harder? trade-off!).
Well-defined problems have specific goals, clearly defined solution paths, and clear expected solutions. Engineering becomes an optimization problem.

Ill-defined problems are those that do not have clear goals, solution paths, or expected solution.

Wicked problems are ill-defined problems that are not understood until after the formulation of a solution [Jeffrey2006]. Design becomes an iterative search problem.

Outlook – Empower Developer

“Shared theories about problem and solution domains that are in the minds of developers” [Peter Naur, 1985]

“Every person’s mental model is incomplete and out of date but we need them all!” [Jessica Kerr, Explore DDD Conference, 2018]

“People are part of the code” [Jean Yang, QCon Panel, 2021]

Flow is a state of mind, a holistic sensation, that people feel when they act with total involvement [Mihaly Csikszentmihalyi]

Clear goals
- Total sense of involvement
- Loss of self-consciousness
- Feeling of control and being in control
- Altered sense of time
- Above average skills and challenges

Thank you for your attention!

Prof. Dr. Holger Giese and Christian Medeiros Adriano
Head of the System Analysis and Modeling Group
Hasso Plattner Institute at the University of Potsdam
# AI in SE: Software Testing

<table>
<thead>
<tr>
<th>Task type</th>
<th>Data type</th>
<th>Dataset</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug detection</td>
<td>code-based data</td>
<td>Linux, MySQL and Apache HTTPD server [12] includes 986 apps, 578 normal apps and 408 vulnerable apps</td>
<td>[IEEE68]</td>
</tr>
<tr>
<td></td>
<td>code-based data</td>
<td></td>
<td>[EL17]</td>
</tr>
<tr>
<td></td>
<td>text-based data</td>
<td>bug report from Mozilla</td>
<td>[EL12]</td>
</tr>
<tr>
<td>Bug localization</td>
<td>code-based data</td>
<td>Defects4J benchmark</td>
<td>[IEEE40]</td>
</tr>
<tr>
<td></td>
<td>code- and text-based data</td>
<td>AspectJ in Bugzilla, SWT, JDT, Tomcat</td>
<td>[IEEE135]</td>
</tr>
<tr>
<td>Vulnerability detection</td>
<td>code-based data</td>
<td>MC&amp;NH dataset</td>
<td>[EL03]</td>
</tr>
<tr>
<td></td>
<td>text-based data</td>
<td>CVE Details websit</td>
<td>[IEEE57]</td>
</tr>
<tr>
<td>Test case generation</td>
<td>code-based data</td>
<td>REAPER</td>
<td>[IEEE17]</td>
</tr>
<tr>
<td></td>
<td>image-based data</td>
<td>MNIST, fashion-MNIST</td>
<td>[IEEE93]</td>
</tr>
<tr>
<td></td>
<td>image-based data</td>
<td>CIFAR-10, CIFAR-100</td>
<td>[IEEE93]</td>
</tr>
<tr>
<td>Program analysis</td>
<td>text-based data</td>
<td>static analysis alarm data</td>
<td>[IEEE62]</td>
</tr>
<tr>
<td>bug classification</td>
<td>code-based data</td>
<td>MozillaProject, Radare2Project</td>
<td>[EL13]</td>
</tr>
</tbody>
</table>

Or suggests a SQL code

```javascript
async function addBook(params) {
    await query(`INSERT INTO books (title, author, isbn, description, image, price, quantity) VALUES ('${params.title}', '${params.author}', '${params.isbn}', '${params.description}', '${params.image}', '${params.price}', '${params.quantity}')`);
}
```

However, accepting this might create a SQL injection vulnerability. The reason is that content of params is passed “as-is” without any protection*

*Four options of avoiding SQL Injection:
1. Use of Prepared Statements (with Parameterized Queries)
2. Use of Properly Constructed Stored Procedures
3. Allow-list Input Validation
4. Escaping All User Supplied Input

https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
Summary of Approach and Results

Chris’ PhD Thesis
Impact of Fault Understanding on bug fixing

Do explanations make bug fixes more accurate?

Yes, but only for explanations that reflect fault understanding

<table>
<thead>
<tr>
<th>Statistic</th>
<th>No Explanation</th>
<th>Root-Cause</th>
<th>Fault removal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>46%</td>
<td>81%</td>
<td>100%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Statistic</th>
<th>No Explanation &lt; Root-cause</th>
<th>Root-Cause &lt; Fault removal</th>
<th>No Explanation &lt; Fault removal</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-value</td>
<td>0.012</td>
<td>0.018</td>
<td>0.000001</td>
</tr>
<tr>
<td>Cohen-d</td>
<td>0.74</td>
<td>0.68</td>
<td>1.48</td>
</tr>
<tr>
<td>Strength</td>
<td>Large</td>
<td>Large</td>
<td>Large</td>
</tr>
<tr>
<td>pos-hoc Power</td>
<td>48%</td>
<td>43%</td>
<td>94%</td>
</tr>
</tbody>
</table>

AI in Software Engineering
Prof. Holger Giese
Hasso Plattner Institute
Summary of Results

1: What are the factors related to the correct identification of a software fault?
Besides programming skill and professional background, the programmer’s confidence and her perceived difficulty of the task comprised to the main factors, i.e., the stronger interventional effects measured by the causal models.

2: How many replications are necessary until we correctly recognize a software fault?
On average 4 times (20% x 20)

3: Are programmers more accurate in their bug fixes if they have access to the explanations?
Yes, programmers are more accurate if they have access to two types of explanations: the root-cause and the fault removal suggestion.

**Required:** Function computing the impact on the utility for each possible rule application

**Open Question:** Can we learn these functions offline (training)?
1) Training
2) Evaluation (meta level)

Real system or Simulator

Adaptation Engine

Managed resource

Analytical Utility Computer

Utility Change Predictor

R Studio

Adaptation Engine

Simulator

Managed resource

Generates prediction models (.pmml)

Predicts impact of adaptation rules

Generates data for machine learning

Observes

Adapts

Simulates

Provides ground truth

Computes impact of adaptation rules

[Ghahremani+2018]
**RQ:** Does the performance approximate the analytic-defined optimum? **YES**

Normalized rewards across prediction models for the combined variant

$$\text{Normalized Reward (mod)} = \frac{\text{Reward (mod)} - \text{Reward (Baseline)}}{\text{Reward (Optimal)} - \text{Reward (Baseline)}}$$
Complex Self-Awareness & Train Goals (4/4)

**Train goals**: adjust goals according to success w.r.t. higher level goals

**PROBLEM**: There is no guarantee that the trained goals are valid due to fact that they always rely on potentially erroneous or outdated measurements/perceptions → optimality is not guaranteed

**Learn runtime models** (known unknowns); parameters, elements, and relations of runtime models are **learned** according to the perception
Some Literature

Books:

- Software Engineering for Self-Adaptive Systems
- Software Engineering for Self-Adaptive Systems II
- Software Engineering for Self-Adaptive Systems III. Assurances

Hasso Plattner Institute