[CS3704] Software Engineering

Shawal Khalid

Virginia Tech

4/1/2024

Announcements

- **Workshop on Friday (4/5)**
  [Workshop] Testing and Development Environments
- **PM3 due this (4/8)**

- **Exam Review Wednesday (4/19) in class**
- **Exam on Friday (4/26)**

Code Analysis

Code Metrics
Software Quality
Code Analysis

Learning Outcomes

By the end of the course, students should be able to:

- **Implement a software system following the software life cycle phases**
- Develop software engineering skills working on a team project
- Identify processes related to phases of the software lifecycle
- Explain the differences between software engineering processes
- Discuss research questions and studies related to software engineering
- Communicate (via demo and writing) details about a developed software application

Warm-Up

Discuss the following with partner or small group:

- Which git command stages changes to the local repository? git add
- What is the difference between the spiral and prototyping model?
- What is discussed in a retrospective? What went well
  What didn't go well
  What to do differently next time
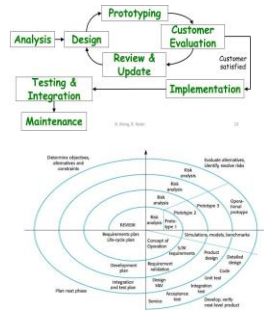
Exam study guide available on the class repository:
<https://github.com/shawalkhalid667/C3704-VT/blob/main/Resources/StudyGuide.md>

Implementation

Goal: translating design into a concrete system (i.e. code)

• Can use any language, but some languages are better suited to certain types of programs than others

• *Software Artifacts*: source code, documentation, configuration files, media, executables, bug database, source code repository, issue trackers…

Why Analyze Code?

Analyzing code is useful for a variety of activities:

- Improving code quality
- Predicting errors
- Optimizing program
- Enforce consistent coding styles
- Measuring complexity
- Evaluating team and process productivity

Software Metrics

Characterize a software system by:

- Assisting in the evaluation and analysis of *design* models;
- Providing an indication of the complexity of *source code*; and
- Facilitate the design of more effective *testing*.

Software Metric Attributes

- Simple and computable
  - Relatively easy to derive and low effort/time for computation
- Empirically and intuitively persuasive
  - Satisfy software engineer's intuitive notions about product
- Consistent and objective
  - Results should be unambiguous
- Consistent in units and dimensions
  - Have desirable mathematical properties and measures
- Programming language independent
  - Analysis based on model and structure of a program
- Effective for high-quality feedback
  - Lead to a higher quality end product!

Basic Code Metrics

- Lines of Code
- Halstead Complexity
- Weighted Methods per Class
- Cyclomatic Complexity
- Inheritance Tree Depth (*object-oriented*)
- Number of Children (*object-oriented*)

Lines of Code

The number of lines of code in a given program.

- Using less code for the same functionality improves readability and maintainability.
  - More lines of code = more maintenance

Lines of Code (cont.)

- **LOC is difficult to measure.**
  - How do we actually count lines of code?
    - i.e. comments, multi-line statements, blanks,…

Example: How many lines of code is this?

```
wc -l
```

5

```
6.1 'wc': Print newline, word, and byte counts
===============================================
'wc' counts the number of bytes, characters, words, and newlines in each
given FILE, or standard input if none are given or for a FILE of '-'.  A
word is a nonzero length sequence of printable characters delimited by
white space.  Synopsis:
```

```
sloc test.py
```

```
----------- Result -----------
              Physical :  5
                Source :  4
               Comment :  0
  Single-line comment :  0
         Block comment :  0
                 Mixed :  0
   Empty block comment :  0
                 Empty :  1
                 To Do :  0

Number of files read :  1

------------------------------
```

simple tool to count SLOC (source lines of code)

```
cloc test.py
```

```
github.com/AlDanial/cloc v 1.88  T=0.01 s (183.1 files/s, 915.7 lines/s)
```

Language	files	blank	comment	code
Python	1	1	0	4

cloc 🔗

Count Lines of Code

Halstead's Metrics

- Metrics of the software should reflect the programming language.
- Calculations:

$\eta = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$

η_1 = the number of distinct operators

η_2 = the number of distinct operands

Length: $N = N_1 + N_2$

N_1 = the total number of operators

N_2 = the total number of operands

Volume: $V = N * \log_2 \eta$

Difficulty: $D = \eta_1 / 2 * N_2 / \eta_2$

Effort: $E = D * V$

```
main()
{
    int a, b, c, avg;
    scanf("%d %d %d", &a, &b, &c);
    avg = (a+b+c)/3;
    printf("avg = %d", avg);
}
```

The distinct *operators* (what action to perform) are:

`main, (), {}, int, scanf, &, =, +, /, printf, ,, ;` (12)

The distinct *operands* (what to apply actions to) are:

`a, b, c, avg, "%d %d %d", 3, "avg = %d"` (7)

Weighted Methods per Class (WMC)

- Metric to indicate the sum of complexities for methods defined in a class.
  - Simple: the # of methods in a class
  - Advanced: Weighted sum

Cyclomatic Complexity

The number of independent paths in a program.

- Also known as McCabe's Cyclomatic Complexity
- Measures the number of linear segments in a method (branches)
- Useful for measuring the *minimum* number of test cases for comprehensive testing
- Calculate:
  - Number of decisions + 1

Cyclomatic Complexity Examples

```
public static String evenOdd(int num) { if (num == 0)
    {
        return "Even";
    } else if (num == 2) { return
        "Even";
    } else if (num == 4) { return
        "Even";
    } else if (num == 6) { return
        "Even";
    } else if (num == 8) { return
        "Even";
    } else {
        return "Odd";
    }
}
```

6

Cyclomatic Complexity Examples…

```java
public static String evenOdd(int num) {
    if (num == 0 || num == 2 || num == 4 || num == 6 || num == 8) {return "Even";

    }
    return "Odd";
}
```

6

```java
public static String evenOdd(int num) {if (num % 2
    == 0) {
        return "Even";
    }
    return "Odd";
}
```
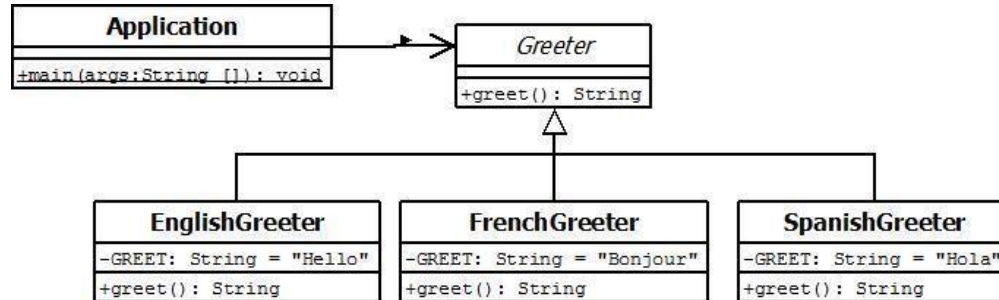
2

Inheritance Tree Depth

- Length from a class node to the root of the class hierarchy tree
    - Number of ancestor classes
    - With multiple inheritance, use the max length from node to root

Number of Children

- Number of direct descendants (subclasses) for each class.

Depth: 2
#Children: 3

Subclasses

- Objects with a large number of children (depth and direct) are:
  - Difficult to modify
  - Require more testing
  - More complex and error-prone
  - **BUT,** have greater reuse of defined methods

Overview

Metric	Desirable Value
Lines of Code (per class)	Lower □
Number of Classes	Higher □
Halstead Complexity	Lower □
Cyclomatic Complexity	Lower □
Weighted Methods Per Class	Lower (tradeoff)
Depth of Inheritance Tree	Low (tradeoff)
Number of Children	Low (tradeoff)

- Indicators for how much effort will be necessary to maintain the class/code.
- Classes with large complexity metrics should often be refactored into two or more classes.

Which Metric?

- Lots of code metrics have been proposed
  - Halstead's measures
  - Cyclomatic complexity
  - Many object-oriented measures
  - …   "The impossible holy grail" **[Fenton]**
- ***But nothing works better (predicting effort and error rate) than counting lines of code (LOC)!***
  - El Emam: "Confounding Effects of Class Size on the Validity of Object-Oriented Metrics" [2001]
  - Herraiz: "Beyond Lines of Code: Do We Need More Complexity Metrics?" [2010]
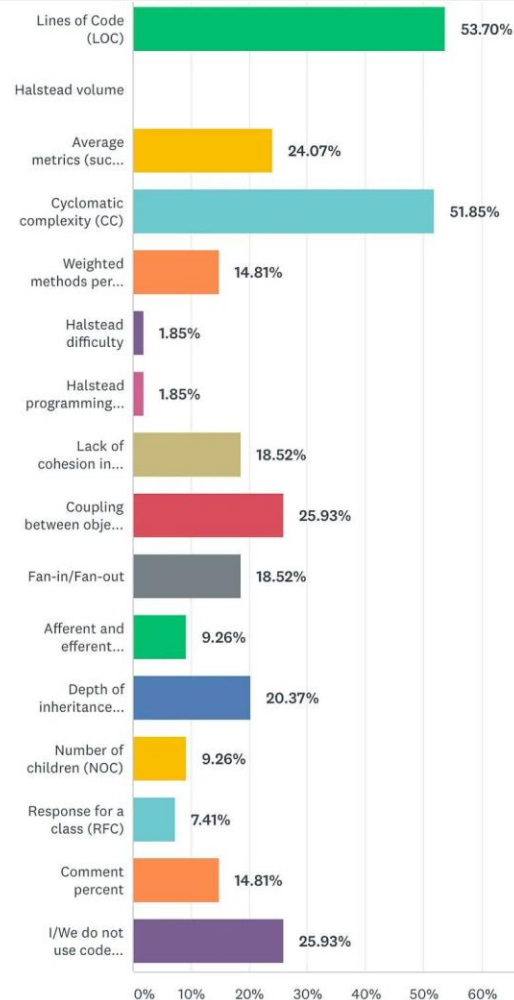
A survey of 78 software engineers found…

Are current metrics sufficient for aspects of SE?

[Sharma]

23
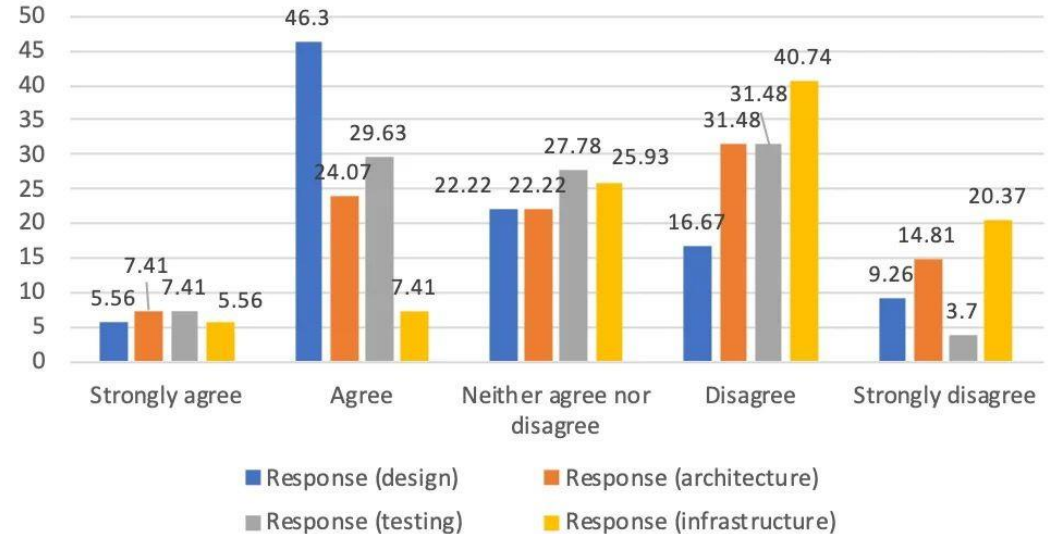
Why Metrics?

Although code metrics are not absolute, they provide a systematic way to assess the quality of software based on a set of clearly defined rules.

- Immediate feedback instead of after-the-fact
- Predict potential problems before deployment
- Recent advancements (i.e. machine learning) have shown promise in generating more useful metrics
  - [Awesome Source Code Analysis Via Machine Learning Techniques](#)

Software Quality

- Ideally, software quality is defined as: *conformance to explicitly stated functional and performance requirements, standards, and implicit characteristics expected in professional development.* [Pressman]

But, what are potential problems with this definition?
[TODO: Discuss with a partner]

- Requirements may change
- Requirements may be incomplete
- Requirements may not be high quality (i.e. ambiguous, unclear, etc.)
- Cannot anticipate the understanding and behavior of users
- Cannot anticipate changes in standards and laws
- Cannot anticipate innovations in technology
- Development processes and practices change
- …

Software Quality (cont.)

Two broad groups for software quality:
1. Factors that can be *directly* measured
    – i.e., coding metrics
2. Factors that can be *indirectly* measured
    – i.e., usability

Code Analysis Tools

- Manual analysis of code is very valuable, but also inefficient.
  - **Ideally, peer code reviews should be combined with automated tools.**
- Many automated tools can programmatically analyze large software systems.

TODO: Discuss any development tools you have used to help with programming.

Static Analysis

The process of automatically examining source code without executing the application.

- analysis tools, code metrics, linters, code clone detection, etc.

Why static analysis?

- Improves code quality
- Finds errors more effectively and efficiently
- Reduces technical debt and software costs
- Increases productivity

How to do static analysis?

- Convert your code into a program representation
  - Text matching
    - Regex, diffs, longest common subsequence, etc.
  - Abstract Syntax Tree (AST)
  - Control Flow Graphs (CFG)
  - Program Dependency Graph
  - Call Graph
  - Tokenization

Abstract Syntax Trees

- Tree representation for the source code structure
  - Node: construct (i.e. statement or loop)
  - Edge: construct relationship
- Created by the compiler at the end of syntax analysis phase
  - Different compilers can define different ASTs

Abstract Syntax Trees (cont.)

```
x := a + b;
y := a * b;
while (y > a) {
    a := a + 1;
    x := a + b
}
```
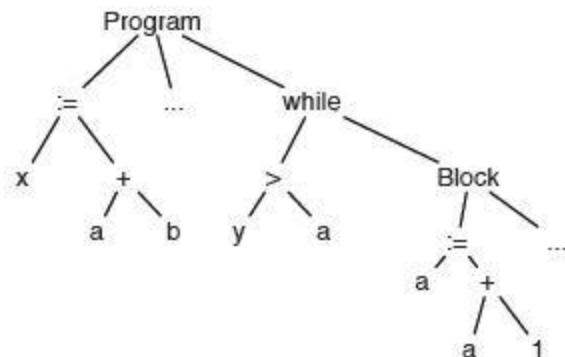
- Make it possible to apply syntax directed translation and transformation.
- Can understand program changes
- Enables automated programming!

Control Flow Graph (CFG)

- Graph notation representation of all paths that might be traversed through a program during its execution.
- Formal Representation:
  CFG = <V, E, Entry, Exit>, where
  – V = vertices or nodes, representing an instruction or basic block (a group of instructions)
  – E = edges, potential flow of control,
  – Entry ∈ V, unique program entry
  – Exit ∈ V, unique program exit
- Basic Block: single entry, single exit
- Program analysis, abstract representation, and basic testing/coverage are built on CFG

Control Flow Graph (cont.)

if-else statement
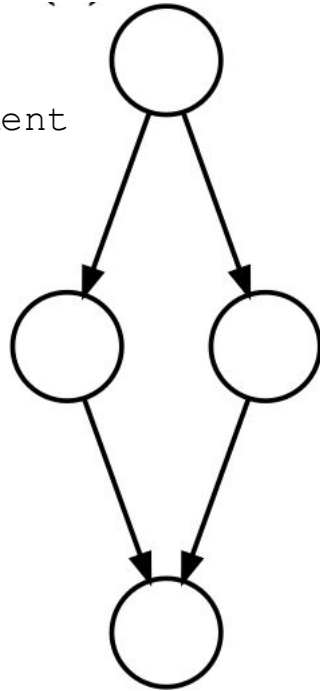
while loop
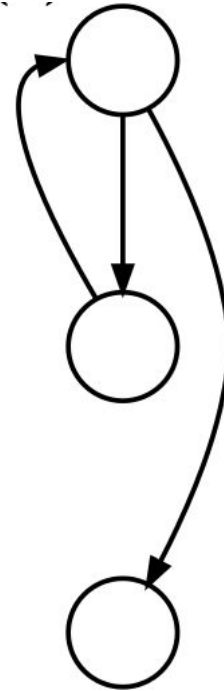
Program Dependency Graph

● A directed graph representing dependencies among code.
  ○ Contain both *control dependence* and *data dependence* edges.

– **Control dependence: A** control depends on **B** if **B**'s execution decides whether or not **A** is executed

– **Data dependence: A** data depends on **B** if **A** uses variable defined in **B**

Program Dependency Graph (cont.)

TODO: Which statement(s) does "`z = x + y;`" depend on? What about "`x = y + x;`"?


```
int x = 1; int y
= 2;
while (x > 0) { x = y
    + x;
}
z = x + y;
```
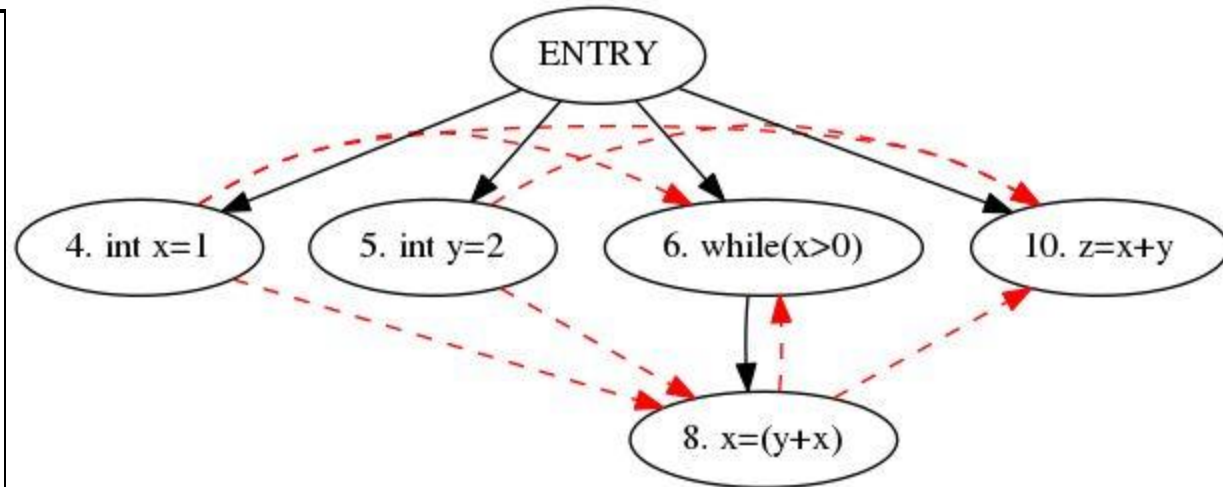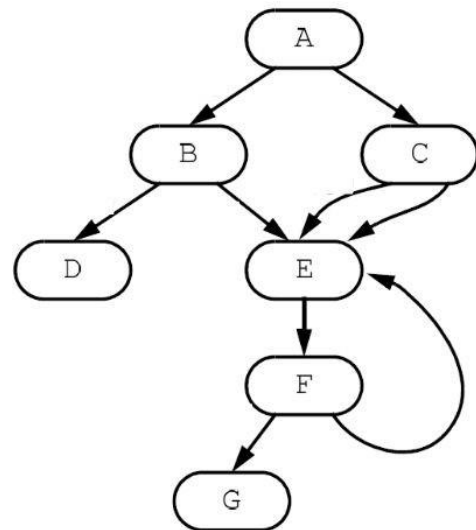
Call Graphs

- A directed graph representing caller-callee relationship between methods/functions
  - Node: methods/functions
  - Edges: calls

Code Clone Detection

- Code fragments in source files that are identical or similar in another.
  - Within a program or across different programs
- **Type 1:** Identical code fragments
  - May have some variations in whitespace, layout, comments…
- **Type 2:** Syntactically equivalent fragments
  - Variations in identifiers, literals, types, etc.
- **Type 3:** Syntactically similar fragments
  - Similar code with inserted, deleted, or modified statements
- **Type 4:** Semantically equivalent
  - But syntactically different code

Code Clone Detection (cont.)

- Text matching
  - Less complex, most widely used
  - Does not take program structure into consideration
  - i.e. regex, diffs, longest common subsequence, etc.
- Token Sequence Matching
  - Type 1 and 2 clones
  - Requires *tokenization*
- Graph Matching
  - Newest, more complex
  - AST, CFG, program dependence graph matching

Tokenization

```
int main(){
    int i = 0;
    static int j=5;
    while(i<20){
        i=i+j;
    }
    std::cout<<"Hello World"<<i<<std::endl;
    return 0;
}
```

Remove white spaces

Tokenization (cont.)

```
int main(){
int i = 0;
static int j=5;
while(i<20){
i=i+j;
}
std::cout<<"Hello World"<<i<<std::endl;
return 0;
}
```

Shorten Names

Tokenization (cont.)

```
int main (){
int i = 0;
int j = 5;
while (i < 20){
i = i + j;
}

cout << "Hello World" << i << endl;
return 0;
}
```

Tokenize literals, and identifiers of types, methods, and variables.

Tokenization

```
$p $p(){
$p $p = $p;
$p $p = $p;
while($p < $p ){
$p = $p + $p;
}
$p << $p << $p << $p;
return $p;
}
```

Dynamic Analysis

The investigation of the properties of a *running* software system.

● Loggers, debuggers, etc.

Why dynamic analysis?

- Gap between run-time and code structure
- Collect runtime execution information
- Finds bugs in application
- Allows for program transformation, optimization
- Modify program behaviors on the fly

How to do dynamic analysis?

- Instrumentation
  - Modify code or runtime to monitor specific components in a system and collect data
- Instrumentation approaches:
    - Source code modification*
    - Byte code modification
    - VM modification

Source Code Modification

● Given a program's source code, how can we modify the code to record which method is called by main() in what order?

```
public class Test {
        public static void main(String[] args) { if (args.length
                == 0) return;
                if (args.length % 2 == 0) printEven(); else
                printOdd();
        }
        public static void printEven() {System.out.println("Even");} public static void
        printOdd() {System.out.println("Odd");}
}
```

Source Code Instrumentation

- Call site instrumentation: Call print(...)before each method call
- Method entry instrumentation: Call print(...) at entry of each method

```
public class Test {
    public static void main(String[] args) {if (args.length
        == 0) return;
        if (args.length % 2 == 0) {System.out.println("printEven() is called");
            printEven();
        } else {
            System.out.println("printOdd() is called");printOdd();
        }
    }
    public static void printEven() {System.out.println("Even");}public static void
    printOdd() {System.out.println("Odd");}
}
```

```
public class Test {
    public static void main(String[] args) {if (args.length
        == 0) return;
        if (args.length % 2 == 0) printEven(); else
        printOdd();
    }
    public static void printEven() {System.out.println("printEven() is
        called");System.out.println("Even");
    }
    public static void printOdd() {System.out.println("printOdd() is
        called");System.out.println("Odd");
    }
}
```

What does all this have to do with SE?

- Not much for day-to-day activities…
- **Development tools!** Many tools use these techniques to automate and support software engineering work.
    - static analysis tools, dynamic analysis tools, debuggers, code optimizers, linters, test runners, CI/CD build tools, memory management, logging, stack trace and error reporting, configuration, test case generation, automated program repair, bots,...
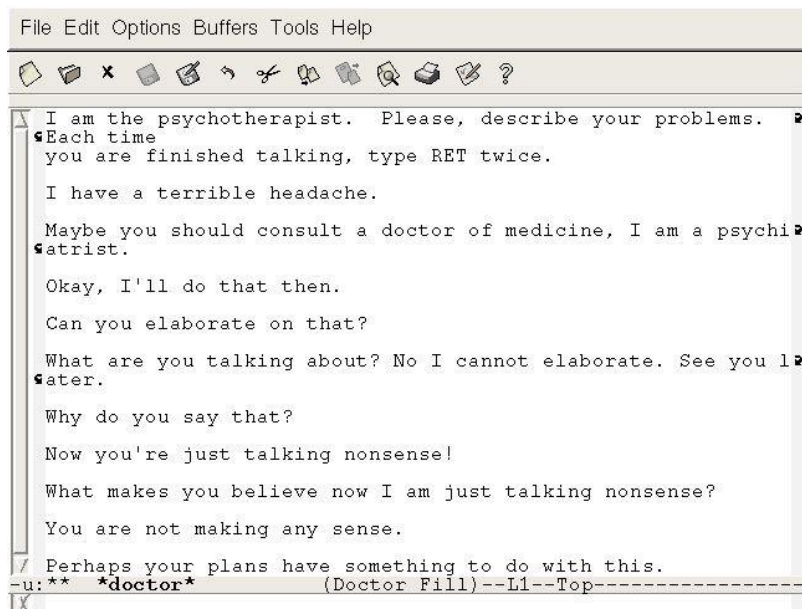
Bots

What is a bot?

● A bot is an agent of automation that can perform automated, repetitive, predefined tasks. Some bots can intelligently adapt their behavior.
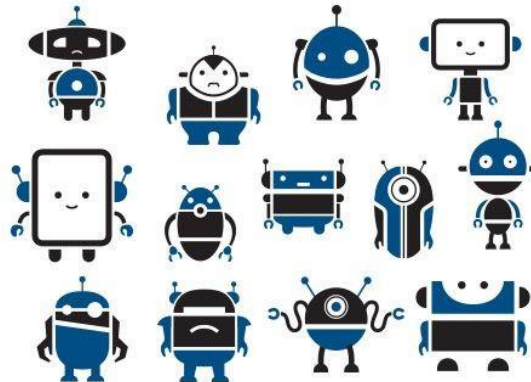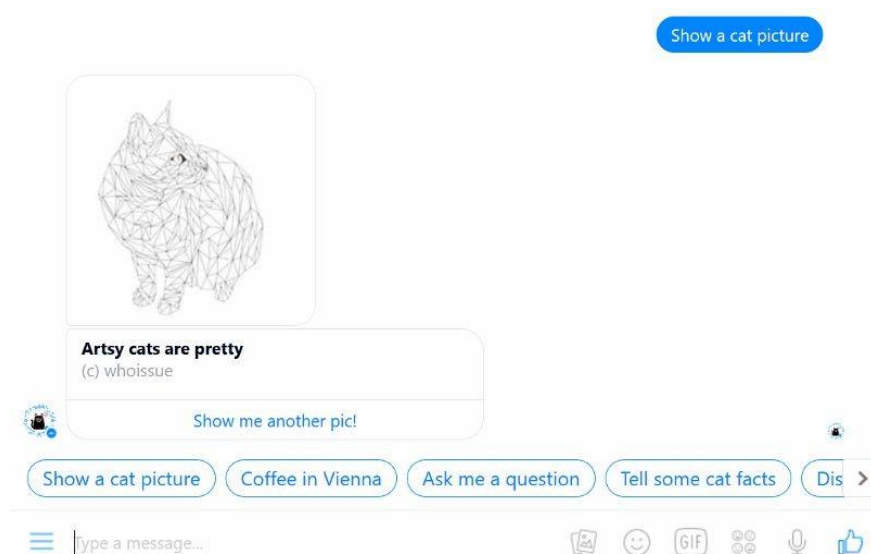
Types of Bots

Chatbots

ELIZA, 1966

Mica the Hipster Cat Bot, 2016

Types of Bots

Devbots

DevBot	Url	Description
Dependabot	https://dependabot.com/	Automatic updates of your dependencies
Greenkeeper	https://github.com/marketplace/greenkeeper	Automatic updates NPM dependencies
Spotbot	https://spotbot.qa/	Takes screenshot of pages and looks for issues
Imgbot	https://imgbot.net/	Automates optimization of images in a project
Deploybot	https://deploybot.com/	Deploys your build
Repairnator	https://github.com/Spirals-Team/repairnator	Automatically repairs build failures on Travis CI
First-timers	https://github.com/apps/first-timers	Creates starter issues for beginners in open source projects
CssRooster	https://huu.la/ai/cssrooster	Writes CSS classes for HTML with Deep learning
Mary-Poppins	https://github.com/mary-poppins	Keeps your merge requests and issues tidy by different plugins
Typot	https://github.com/chakki-works/typot	Fixes typos in merge requests
Marbot	https://marbot.io/	Manage CloudWatch alerts

Defining and Classifying Software Bots: A Faceted Taxonomy
http://chisel.cs.uvic.ca/pubs/lebeuf-BotSE2019.pdf

Current and Future Bots in Software Development
https://github.com/CSC-510/Course/blob/master/Materials/p7-erlenhov.pdf

"An ideal DevBot is an artificial software developer which is autonomous, adaptive, and has technical as well as social competence."

Types of Bots

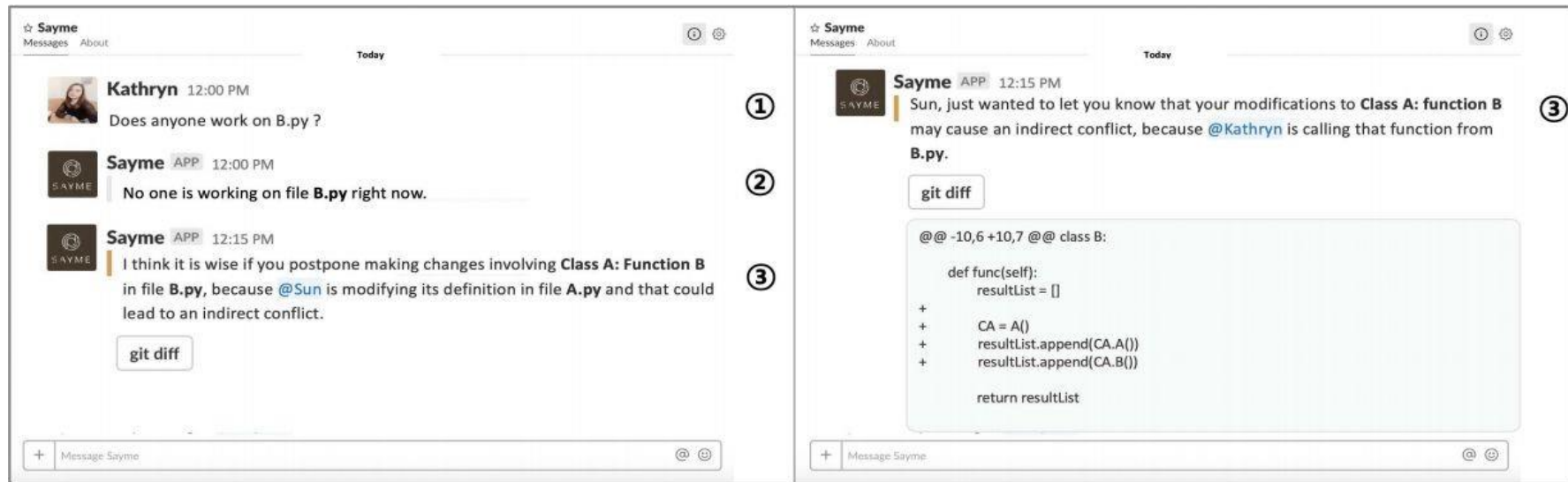## **Chat-DevBots?** Mediate between humans

Fig. 2. Example of interaction with Sayme (indirect conflict).

Bot Design Patterns

- **Notifiers**: *Only send messages (typically scheduled).*
- **Reactors:** *Respond to events; no memory.*
- **Space Reactors:** *Customize reaction based on space/context (room/channel).*
- **Responders:** *Respond to events; maintains memory, knows user.*
- **Space Responders:** *Responders with memory of events in a space/context.*
- **Conversationists:** *Maintains full conversation context.*
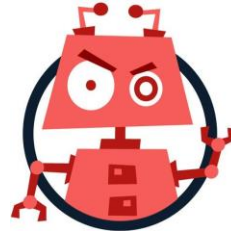
Bots vs. Humans

53
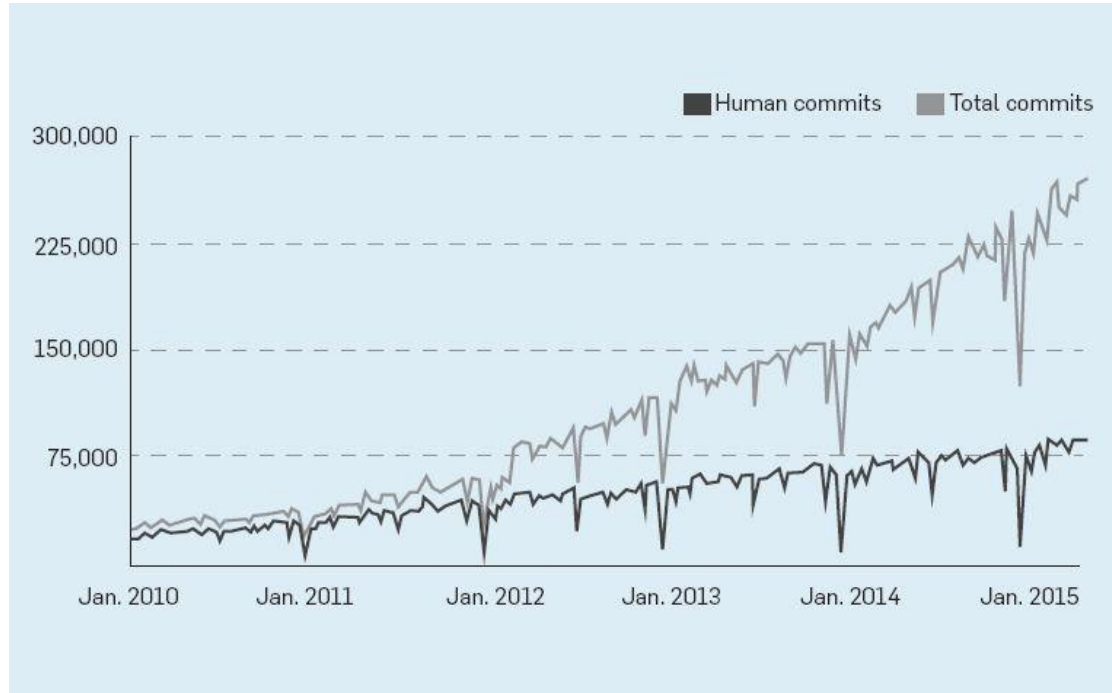
Risks and Issues Using Bots
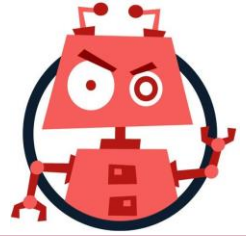
What are some prospective problems with using bots in software engineering? (HW4)

Risks and Issues (cont.)

Error Prone Static Analysis Tool #82

Open **cass-green** wants to merge 1 commit into `apache:master` from `cass-green:master`

💬 Conversation **0**   ⦿ Commits **1**   ☑ Checks **0**   ⬡ Files changed **1**

Social Context   **Developer Workflow**

cas

Looks like you're not using any error-checking in your Java build. This pull requests adds a static analysis tool
Error Prone, created by Google to find common errors in Java code. For example, running `mvn compile` on
the following code:

```
public boolean validate(String s) {
        return s == this.username;
}
```

would identify this error:

```
[ERROR] src/main/java/HelloWorld.java:[17,17] error: [StringEquality] String comparis
[ERROR]      (see https://errorprone.info/bugpattern/StringEquality)
```

If you think you might want to try out this plugin, you can just merge this pull request. Please feel free to add
any comments below explaining why you did or did not find this recommendation useful.

V. RESULTS

A. *Bot Effectiveness*

Out of 52 recommendations, only *two* were accepted by developers. The remaining were categorized as ineffective recommendations due to any response. aluation.

	n	**Percent**
Merged	2	4%
Closed	10	19%
No Response	40	77%

TABLE I: Pull Request Results

An overwhelming 96% of *tool-recommender-bot* recommendations were ineffective. Of the 12 recommendations that did receive developer feedback, 83% were rejected by developers who closed the pull request. Two recommendations were merged, however in one case another contributor created a GitHub issue because our pull request caused problems with the project build. The changes made by the bot were then reverted in a later pull request, removing ERROR PRONE from the project. Even though the tool was eventually removed, we still categorize this as an effective recommendation because the developers accepted the pull request to try the tool.

Next Time…

- **Implementation & Maintainance**

- **Study Guide: main/resources/StudyGuide.md**

References

- Dr. Chris Brown
- RS Pressman. *"Software engineering: a practitioner's approach"*.
- Tushar Sharma, et al. "*Do We Need Improved Code Quality Metrics?*", 2020
- Rachel Potvin, et al. *"Why Google Stores Billions of Lines of Code in a Single Repository"*. 2016
- Chris Brown, et al. *"Sorry to Bother You: Designing Bots for Effective Recommendations"*. 2019
- Na Meng and Barbara Ryder
- Chris Parnin, Sarah Heckman