# CS 563: Software Maintenance And Evolution

# **Software Defect Prediction**

Oregon State University, Spring 2024

# Today's plan

- Learn about how to brainstorm and solve a problem using *software defect prediction* as an example
- In-class exercise: Brainstorm ideas for your own class projects

# Software Defect Prediction

- What are software defects?

# Software Defect Prediction

- What are software defects?
  - *Bugs or errors in the code, documentation, requirements, etc. where the actual behavior or software differs from its expected behavior*

# Software Defect Prediction

- What are software defects?
  - *Bugs or errors in the code, documentation, requirements, etc. where the actual behavior or software differs from its expected behavior*
- What is software defect prediction and why should we care about predicting software defects?

# Software Defect Prediction

- What are software defects?
  - *Bugs or errors in the code, documentation, requirements, etc. where the actual behavior or software differs from its expected behavior*

- What is software defect prediction and why should we care about predicting software defects?
  - A real-world software consists of many components
  - Testing all components in every release/update is time-consuming
  - What if there was a technique that tells us about the components that are more likely to have defects in the current release?
  - This would:
    - Increase the productivity of software engineers
    - Increase the quality of the product
    - Reduce the testing costs

# Software Defect Prediction

- How to know which software modules are more likely to be defective (or buggy)?

# Software Defect Prediction

- How to know which software modules are more likely to be defective (or buggy)?

Modules of previous versions of the software

| Module Id | x1 = branch count | x2 = LOC | x3 = halstead | ... | y = defective ? |
|---|---|---|---|---|---|
| 1 | 18 | 1000 | 1 | ... | No |
| 2 | 30 | 900 | 10 | ... | Yes |
| 3 | 20 | 5000 | 3 | ... | Yes |
| ... | ... | ... | ... | ... | ... |

→ Machine Learning Algorithm →

AI Seminar on AI4SE: Opportunities and Challenges

- https://engineering.oregonstate.edu/events/ai-seminar-ai-software-engineering-ai4se-opportunities-and-challenges
- https://media.oregonstate.edu/media/t/1_jqu7qqrc/232665543

# Software Defect Prediction

- How to know which software modules are more likely to be defective (or buggy)?

Modules of previous versions of the software

| Module Id | x1 = branch count | x2 = LOC | x3 = halstead | ... | y = defective ? |
|---|---|---|---|---|---|
| 1 | 18 | 1000 | 1 | ... | No |
| 2 | 30 | 900 | 10 | ... | Yes |
| 3 | 20 | 5000 | 3 | ... | Yes |
| ... | ... | ... | ... | ... | ... |

→ Machine Learning Algorithm →

New module **x** for new version of the software →

→ Yes/No

# Software Defect Prediction

**Recap:**

- What problem are we solving? (Problem statement)
- Why should one care about solving that problem? (Motivation)
- How are we solving it? (Approach)
  - **High-level idea:** use ML to predict the modules/components of software that are more likely to be buggy
  - **Assumptions?**

# Software Defect Prediction

**Recap:**

- What problem are we solving? (Problem statement)
- Why should one care about solving that problem? (Motivation)
- How are we solving it? (Approach)
  - **High-level idea:** use ML to predict the modules/components of software that are more likely to be buggy
  - **Assumptions:**
    - Multiple versions/releases of software exist; You have access to the data that tells you defects encountered and fixed in the past.
    - What if above doesn't hold?

# Software Defect Prediction

**Recap:**

- What problem are we solving? (Problem statement)
- Why should one care about solving that problem? (Motivation)
- How are we solving it? (Approach)
  - **High-level idea:** use ML to predict the modules/components of software that are more likely to be buggy
  - **Assumptions:**
    - Multiple versions/releases of software exist; You have access to the data that tells you defects encountered and fixed in the past.
    - What if above doesn't hold?
      - Find "similar" software application and predict across software

# Software Defect Prediction

**Refining Approach:**

- How are we solving it? (Approach)
  - **High-level idea:** use ML to predict the modules/components of software that are more likely to be buggy
  - **Assumptions:**
    - Multiple versions/releases of software exist; You have access to the data that tells you defects encountered and fixed in the past.
    - What if above doesn't hold?
      - Find "similar" software application and predict across software
  - What else do we need to think about to concretize the high-level idea?

# Software Defect Prediction

**Refining Approach:**

- How are we solving it? (Approach)
  - **High-level idea:** use ML to predict the modules/components of software that are more likely to be buggy
  - **Assumptions:**
    - Multiple versions/releases of software exist; You have access to the data that tells you defects encountered and fixed in the past.
    - What if above doesn't hold?
      - Find "similar" software application and predict across software
  - **What else do we need to think about to concretize the high-level idea?**
    - Input features

# Software Defect Prediction

**Refining Approach:**

- How are we solving it? (Approach)
  - **High-level idea:** use ML to predict the modules/components of software that are more likely to be buggy
  - **Assumptions:**
    - Multiple versions/releases of software exist; You have access to the data that tells you defects encountered and fixed in the past.
    - What if above doesn't hold?
      - Find "similar" software application and predict across software
  - **What else do we need to think about to concretize the high-level idea?**
    - Input features
    - Machine Learning algorithm

These two will:
- demonstrate the novelty of your solution
- Impact the effectiveness of your solution

# Software Defect Prediction

**Refining Approach:**

- How are we solving it? (Approach)
  - **High-level idea:** use ML to predict the modules/components of software that are more likely to be buggy
  - **What input features can we use?**
    - What is the **key insight**?

# Software Defect Prediction

**Refining Approach:**

- How are we solving it? (Approach)
  - **High-level idea:** use ML to predict the modules/components of software that are more likely to be buggy
  - **What input features can we use?**
    - What is the **key insight**?

      The more "complex" is the code, the more likely it is to be buggy or defective.

# Software Defect Prediction

**Refining Approach:**

- How are we solving it? (Approach)
  - **High-level idea:** use ML to predict the modules/components of software that are more likely to be buggy
  - **What input features can we use?**
    - What is the **key insight**?

      The more "complex" is the code, the more likely it is to be buggy or defective.
    - How do we measure **"code complexity"**?

# Software Defect Prediction

**Refining Approach:**

- How are we solving it? (Approach)
  - **High-level idea:** use ML to predict the modules/components of software that are more likely to be buggy
  - **What input features can we use?**
    - What is the **key insight**?

      The more "complex" is the code, the more likely it is to be buggy or defective.
    - How do we measure **"code complexity"**?
      - Lines of code (LOC)
        - Commented lines
        - Non-commented lines
      - McCabe cyclomatic complexity (control flow)
      - Halstead complexity measures (operators and operands)

# How to measure code complexity?

**McCabe cyclomatic complexity:** measures complexity by counting the number of linearly independent paths in the control flow graph of the code.

- We will have more possible paths if we have more condition statements in our code.
- McCabe cyclomatic complexity = number of simple conditions + 1.

# How to measure code complexity?

**McCabe cyclomatic complexity:** measures complexity by counting the number of linearly independent paths in the control flow graph of the code.

- We will have more possible paths if we have more condition statements in our code.
- McCabe cyclomatic complexity = number of simple conditions + 1.
- Simple conditions are conditional statements without OR or AND.
- E.g.,:
  - If (a > b) —> this counts as one simple condition
  - While (a > b) —> this counts as one simple condition
  - For (a=b; a > b; b++) —> this counts as one simple condition
  - Do…while (a > b) —> this counts as one simple condition

# How to measure code complexity?

**McCabe cyclomatic complexity:** measures complexity by counting the number of linearly independent paths in the control flow graph of the code.

- For compound conditions, count each simple condition inside it.
- If (a > b) OR (a > 2) —> this counts as **two** simple conditions

```
if (a > b) OR (a > 2)
    statement
```

```
if (a > b)
    statement
else if (a > 2)
    statement
```

# How to measure code complexity?

**McCabe cyclomatic complexity:** measures complexity by counting the number of linearly independent paths in the control flow graph of the code.

- For compound conditions, count each simple condition inside it.
- If (a > b) OR (a > 2) —> this counts as **two** simple conditions
- If (a > b) AND (a > 2) —> this counts as **two** simple conditions

```
if (a > b) AND (a > 2)
    statement
```

```
if (a > b)
    if (a > 2)
        statement
```

# McCabe cyclomatic complexity Quiz (1/3)

**McCabe cyclomatic complexity:** measures complexity by counting the number of linearly independent paths in the control flow graph of the code.

McCabe cyclomatic complexity = number of simple conditions + 1.

- What is the McCabe cyclomatic complexity for the following code snippets?

```
int a = 1;
int b = 2;
int c = a + b;
```

```
int a = 1;
int b = 2;
if (a > b)
    a = b;
int c = a + b;
```

```
int a = 1;
int b = 2;
if (a > b)
    a = b;
else
    b = a;
int c = a + b;
```

# McCabe cyclomatic complexity Quiz (1/3)

**McCabe cyclomatic complexity:** measures complexity by counting the number of linearly independent paths in the control flow graph of the code.

McCabe cyclomatic complexity = number of simple conditions + 1.

- What is the McCabe cyclomatic complexity for the following code snippets?

```
int a = 1;
int b = 2;
int c = a + b;
```

McCabe Complexity = 1

```
int a = 1;
int b = 2;
if (a > b)
    a = b;
int c = a + b;
```

McCabe Complexity = 2

```
int a = 1;
int b = 2;
if (a > b)
    a = b;
else
    b = a;
int c = a + b;
```

McCabe Complexity = 2

# McCabe cyclomatic complexity Quiz (2/3)

**McCabe cyclomatic complexity:** measures complexity by counting the number of linearly independent paths in the control flow graph of the code.

McCabe cyclomatic complexity = number of simple conditions + 1.

- What is the McCabe cyclomatic complexity for the following code snippets?

```
int a = 1;
int b = 2;
if (a > b && a > 1)
    a = b;
int c = a + b;
```

```
int a = 1;
int b = 2;
if (a > b || a > 1)
    a = b;
int c = a + b;
```

```
int a = 1;
int b = 2;
if (a > b)
    a = b;
if (2 * a > b)
    a = b;
int c = a + b;
```

# McCabe cyclomatic complexity Quiz (2/3)

**McCabe cyclomatic complexity:** measures complexity by counting the number of linearly independent paths in the control flow graph of the code.

McCabe cyclomatic complexity = number of simple conditions + 1.

- What is the McCabe cyclomatic complexity for the following code snippets?

```
int a = 1;
int b = 2;
if (a > b && a > 1)
    a = b;
int c = a + b;
```

McCabe Complexity = 3

```
int a = 1;
int b = 2;
if (a > b || a > 1)
    a = b;
int c = a + b;
```

McCabe Complexity = 3

```
int a = 1;
int b = 2;
if (a > b)
    a = b;
if (2 * a > b)
    a = b;
int c = a + b;
```

McCabe Complexity = 3

# McCabe cyclomatic complexity Quiz (3/3)

**McCabe cyclomatic complexity:** measures complexity by counting the number of linearly independent paths in the control flow graph of the code.

McCabe cyclomatic complexity = number of simple conditions + 1.

- What is the McCabe cyclomatic complexity for the following code snippets?

```
int a = 1;
int b = 2;
while (a > b)
    a--;
int c = a + b;
```

```
switch (a) {
    case 1:
        a += 10;
        break;
    case 2:
        a += 30;
        break;
    case 3:
        a += 60;
        break;
    default:
        a += 1;
        break;
}
int c = a + b;
```

```
int a = 1;
int b = 2;
int c = 3;
try {
    a= 10;
} catch (ExceptionType1 name) {
    b = 10;
} catch (ExceptionType2 name) {
    c = 10;
}
```

# McCabe cyclomatic complexity Quiz (3/3)

**McCabe cyclomatic complexity:** measures complexity by counting the number of linearly independent paths in the control flow graph of the code.

McCabe cyclomatic complexity = number of simple conditions + 1.

- What is the McCabe cyclomatic complexity for the following code snippets?

```
int a = 1;
int b = 2;
while (a > b)
    a--;
int c = a + b;
```

McCabe Complexity = 2

```
switch (a) {
    case 1:
        a += 10;
        break;
    case 2:
        a += 30;
        break;
    case 3:
        a += 60;
        break;
    default:
        a += 1;
        break;
}
int c = a + b;
```

McCabe Complexity = 4

```
int a = 1;
int b = 2;
int c = 3;
try {
    a= 10;
} catch (ExceptionType1 name) {
    b = 10;
} catch (ExceptionType2 name) {
    c = 10;
}
```

McCabe Complexity = 3

# How to measure code complexity?

**Halstead Complexity Measures:** measures complexity based on operators and operands used in the code.

- n1 = number of distinct operators (!=, !, %, /, *, +, &&, ||, etc.)
- n2 = number of distinct operands (identifiers that are not reserved words, constants (character, number, or string constants), types (bool, char, double), etc.
- N1 = total number of the operators
- N2 = total number of operands

# How to measure code complexity?

**Halstead Complexity Measures:** measures complexity based on operators and operands used in the code.

- n1: = number of distinct operators
- n2 = number of distinct operands
- N1 = total number of the operators
- N2 = total number of operands

- **Code vocabulary**: n = n1 + n2
- **Code length**: N = N1 + N2
- **Volume**: $V = N \log_2 n$
- **Difficulty**: D = n1 / 2 * N2 / n2
- ...

As the number of unique operators and operands in a program increases, the complexity of understanding and maintaining the program also increases. The logarithm in the formula is used to reflect this non-linear relationship. The base of the logarithm (2 in this case) is chosen because it reflects the binary nature of computation in computers (0s and 1s).

# How to measure code complexity?

**Halstead Complexity Measures:** measures complexity based on operators and operands used in the code.

- n1: = number of distinct operators
- n2 = number of distinct operands
- N1 = total number of the operators
- N2 = total number of operands

- **Code vocabulary**: n = n1 + n2
- **Code length**: N = N1 + N2
- **Volume**: $V = N \log_2 n$
- **Difficulty**: D = n1 / 2 * N2 / n2
- …

The difficulty metric reflects the mental effort required to understand the relationships and actions within a program. A lower difficulty score indicates that the program is easier to understand, while a higher difficulty score suggests that the program is more complex and harder to comprehend.

# How to measure code complexity?

## Lines of Code (LOC)

- **Actual executable lines of code** – higher number means more complex code
- **Lines of code vs Statements of code**?
- **Code comments** – higher number means easy to comprehend the code and therefore lower complexity?

# How to measure code complexity?

## Lines of Code (LOC)

- **Actual executable lines of code** – higher number means more complex code
- **Lines of code vs Statements of code**?
- **Code comments** – higher number means easy to comprehend the code and therefore lower complexity?

## What about process-centric (instead of code-centric) metrics?

| Metric name | Definition |
|---|---|
| REVISIONS | Number of revisions of a file |
| REFACTORINGS | Number of times a file has been refactored[1] |
| BUGFIXES | Number of times a file was involved in bug-fixing[2] |
| AUTHORS | Number of distinct authors that checked a file into the repository |
| LOC_ADDED | Sum over all revisions of the lines of code added to a file |
| MAX_ LOC_ADDED | Maximum number of lines of code added for all revisions |
| AVE_LOC_ADDED | Average lines of code added per revision |
| LOC_DELETED | Sum over all revisions of the lines of code deleted from a file |
| MAX_ LOC_DELETED | Maximum number of lines of code deleted for all revisions |
| AVE_ LOC_DELETED | Average lines of code deleted per revision |
| CODECHURN | Sum of (added lines of code – deleted lines of code) over all revisions |
| MAX_ CODECHURN | Maximum CODECHURN for all revisions |
| AVE_ CODECHURN | Average CODECHURN per revision |
| MAX_CHANGESET | Maximum number of files committed together to the repository |
| AVE_CHANGESET | Average number of files committed together to the repository |
| AGE | Age of a file in weeks (counting backwards from a specific release) |

# Software Defect Prediction

**Recap:**

- How are we solving it? (Approach)
  - **High-level idea:** use ML to predict the modules/components of software that are more likely to be buggy
  - **Assumption:**
    - Multiple versions/releases of software exist; You have access to the data that tells you defects encountered and fixed in the past.
    - What if above doesn't hold?
      - Find "similar" software application and predict across software
  - **What else do we need to think about to concretize the high-level idea?**
    - Input features
    - Machine Learning algorithm

Now that we have some ideas of input features, lets talk about the second aspect of the approach

# What machine learning approach to consider?

- **Option-1**: supervised learning technique

- **Option-2**: unsupervised learning technique

# What machine learning approach to consider?

- **Option-1**: supervised learning technique
  - Requires a labelled training data
  - Shown to be very effective for classification problems
  - E.g., Naive baise, Support Vector Machine, Neural Networks
  - Pros and Cons?
- **Option-2**: unsupervised learning technique
  - Does not require any training dataset
  - E.g.?

# What machine learning approach to consider?

- **Option-1**: supervised learning technique
  - Requires a labelled training data
  - Shown to be very effective for classification problems
  - E.g., Naive baise, Support Vector Machine, Neural Networks
  - Pros and Cons?
- **Option-2**: unsupervised learning technique
  - Does not require any training dataset
  - E.g. K-means, agglomerative clustering, genetic algorithm, etc.
  - Pros and Cons?
- Which option would you choose and why?
- How will you deal with the **class-imabalance problem?**

# How to evaluate your solution?

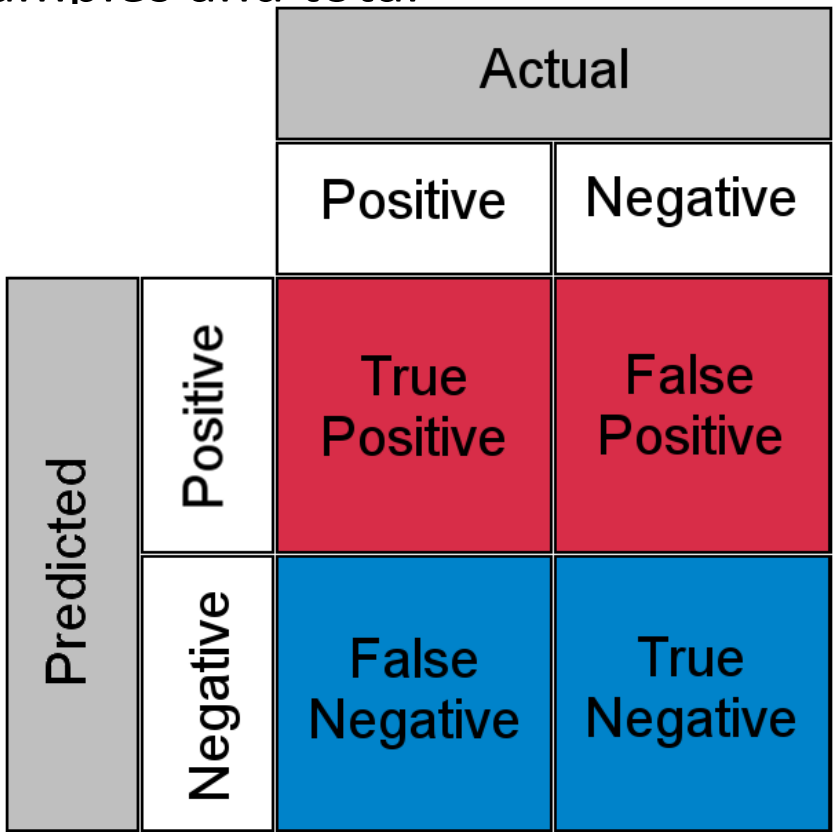
- **Dataset:**
  - Real-world open-source software?
  - Existing evaluations exist?

# How to evaluate your solution?

- **Evaluation Metrics:**
  - How do we measure the effectiveness of a classifier?
    - **Precision**: ratio of correctly classified positive examples and total examples
    - **Recall/ True positive rate (TPR) / Sensitivity**: ratio of positive examples that are correctly identified
    - **False positive rate (FPR)**: ratio of negative examples that are incorrectly classified
    - **Specificity = 1 – FPR**: ability of the model to correctly identify negative instances

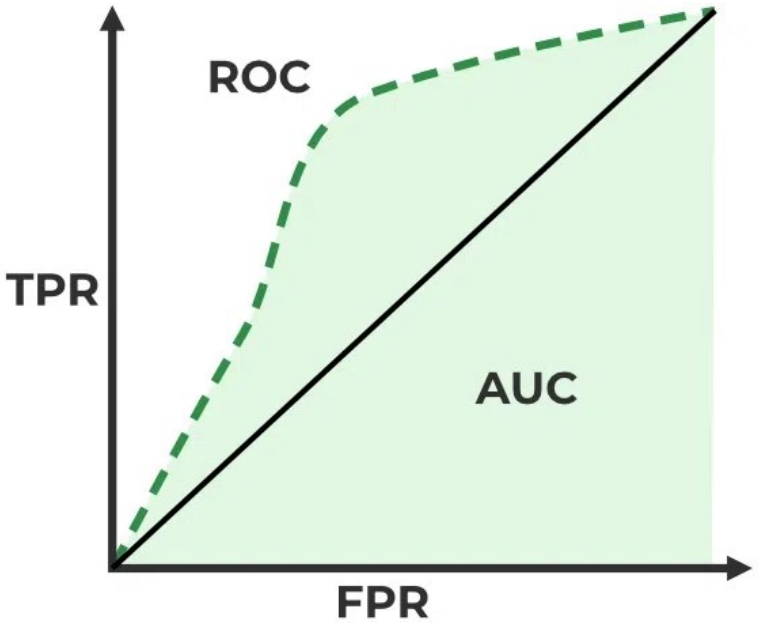
# How to evaluate your solution?

- **Evaluation Metrics:**
  - How do we measure the effectiveness of a classifier?
    - **Precision**: ratio of correctly classified positive examples and total examples
    - **Recall/ True positive rate (TPR) / Sensitivity**: ratio of positive examples that are correctly identified
    - **False positive rate (FPR)**: ratio of negative examples that are incorrectly classified
    - **Specificity = 1 – FPR**: ability of the model to correctly identify negative instances
    - **Confusion matrix**



# How to evaluate your solution?

- **Evaluation Metrics:**
  - How do we measure the effectiveness of a classifier?
    - **Precision** = TP / (TP + FP)
    - **Recall/ True positive rate** (TPR) / Sensitivity
      = TP / (TP + FN)
    - **False positive rate** (FPR)
      = FP / (FP + TN)
    - **Specificity**
      = 1 – FPR
    - **Confusion matrix**
    - **Receiver operating characteristic (ROC) curve**
    - **Area under the curve (AUC)**

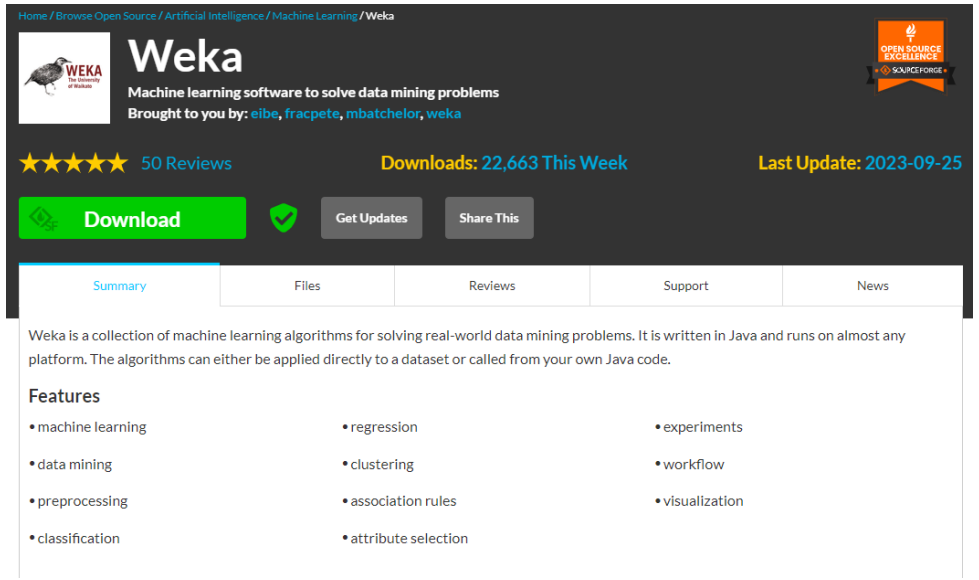


# How to evaluate your solution?

- **Dataset:**
  - Real-world open-source software
  - Existing evaluations exist
- **Evaluation Metrics:**
  - **How do we measure the effectiveness of a classifier?**
    - Precision
    - Recall
    - True positive rate
    - False positive rate
    - Confusion matrix
    - AUC ROC curve
- **Implementation?**

# How to evaluate your solution?

- **Dataset:**
  - Real-world open-source software
  - Existing evaluations exist
- **Evaluation Metrics:**
  - **How do we measure the effectiveness of a classifier?**
    - Precision
    - Recall
    - True positive rate
    - False positive rate
    - Confusion matrix
    - AUC ROC curve
- **Implementation?**



# Software Defect Prediction

**What should we do next now that we have some idea of solving the problem?**

- Literature Survey!
  - What all features and ML techniques have existing solutions tried?
  - What kind of datasets and metrics are used for evaluation? What are their limitations?

- **High-level idea:** use ML to predict the modules/components of software that are more likely to be buggy
- **Assumption:**
  - Multiple versions/releases of software exist; You have access to the data that tells you defects encountered and fixed in the past.
  - What if above doesn't hold?
    - Find "similar" software application and predict across software
- **What else do we need to think about to concretize the high-level idea?**
  - Input features
  - ML algorithm (supervised or unsupervised)

These two will:
- demonstrate the novelty of your solution
- Impact the effectiveness of your solution

# Announcements

- Project idea proposal assignment released
- Homework-1 assignment released
- Paper selection assignment released
  - A list of 10 papers associated with their presentation date.
  - A group of at most 2 students will present each paper (selection is on the FCFS basis). Read the assignment for more details.
  - Everyone will write and submit a review of all the papers individually (due before that paper's presentation date)
  - Form a group (optional) and use the technique to read papers we discussed to go through all papers. Discuss papers with your partner and select the one that interests you.
  - NOTE: Upcoming lectures will cover the concepts described in the research papers, so don't reject a paper because you feel you don't know the specific concepts used in that paper. Ask me, if you have any doubts about whether a specific topic will be covered or not.

# In-class exercise: brainstorm project ideas

- Either individually or in a group of 2, think about:
  - The problem that you would like to solve (problem statement)
  - Why should one care about solving that problem (motivation)
  - Some high-level approach to solve the problem
  - MAKE NOTES!