Understanding Your Code in a More Cost-Effective Way

W. Eric Wong
Department of Computer Science
The University of Texas at Dallas
ewong@utdallas.edu
http://www.utdallas.edu/~ewong

Speaker Biographical Sketch

• Professor & Director of International Outreach
  Department of Computer Science
  University of Texas at Dallas

• Guest Researcher
  Computer Security Division
  National Institute of Standards and Technology (NIST)

• Vice President, IEEE Reliability Society
• Secretary, ACM SIGAPP (Special Interest Group on Applied Computing)
• Principal Investigator, NSF TUES (Transforming Undergraduate Education in Science, Technology, Engineering and Mathematics) Project
  – Incorporating Software Testing into Multiple Computer Science and Software Engineering Undergraduate Courses

• Founder & Steering Committee co-Chair for the SERE conference
  (IEEE International Conference on Software Security and Reliability)
  (http://paris.utdallas.edu/sere13)
PART I

Locating Program Features using Execution Slices


Click on the following pdf icon to download the paper
What is a Feature

- A feature is an abstract description of a functionality given in the specification
  - Example: the ATM software has three features
    - Withdraw
    - Deposit
    - Balance
  - Example: the UNIX wordcount program has three features
    - Number of lines, words, characters

Do You Understand Customer Feedback

- Requirement changes and enhancement requests are usually specified in terms of features affected, not in terms of software components that must be modified
- Example: A telephone switch
  - Call Setup
  - Call Waiting
  - Speed Dialing
Here is How

- Software developers must locate and understand the code associated with the affected features before they can translate change requests into code change.

Features mapping Software components

Software Structure from the Program Feature Point of View

- Well-designed Software Systems
  - A high degree of cohesion
    - A cohesive module should ideally do just one thing
  - A low degree of coupling
  - Each module addresses a specific subfunction of the requirements and has a simple interface when viewed from other parts of the program structure
  - A clear mapping between each feature and its corresponding code segments

- Software systems in the real world
  - Low cohesion & high coupling
  - Program features are mixed together in the code – across modules which are seemingly unrelated
**A Challenge**

- In a complex software system it is not unusual to find that modifications made to one feature, which can be viewed as a functionality of the system, have *adverse impacts* on other *seemingly unrelated features*.
- Such impacts can subsequently change the behavior of those features and *cause a system failure*.
- Need a **good understanding** of the system.

**Objective**

- Locate program code relevant to a particular feature in order to provide software programmers and maintainers with a good *starting point* for quick program understanding.
  - Develop novel heuristics and experiment with them to identify
    - Code unique to the given feature
    - Code common to the given feature and others
  - Examine factors which affect the code so identified.
Three Different Approaches

- Systematic
  - Provides a good understanding
  - Impractical for large complicated systems

- As-needed
  - Less expensive and less time-consuming
  - Miss some non-local interactions between features

- Execution Slice-based
  - An execution slice is the set of program components (blocks, decisions, c-uses, or p-uses) executed by a test input

- Qualitative description versus quantitative measurement

Reading Documentation Does Not Work

- Does not exist
- Incomplete and difficult to understand
- Not updated
- Implementation spread across several non-adjacent modules
- Do not want to read it
A Factor to Consider

- Most software development projects have a set of regression tests to help find bugs in the next release. But these tests can be used for more than just finding errors.

A Better Strategy

- Instead of spending time trying to understand the system, let the system tell you how it works.
**Execution Slice-Based Technique**

Code identified depends on which *heuristic* is applied, which by itself is affected not only by how *invoking and excluding tests* are selected but also how the *features are implemented* in the program.

---

**Definitions: Invoking & Excluding Tests**

- For a given program $P$ and a feature $F$
  - An invoking test is a test that when executed on $P$ shows the functionality of $F$.
  - Example
    - $P$: The wordcount program (wc) 
    - $F$: The functionality to count the *number of characters*
      - `'wc -c data'` (say $t_1$) is an invoking test
  - An excluding test is one that does not
    - `'wc -w data'` is an excluding test
  - An invoking test is focused on $F$ if it exhibits only $F$ and no other features
    - $t_1$ is a focused invoking test
    - *`wc data`* is NOT because in addition to the number of characters, it also returns the number of words and lines
**Number of Tests Required**

- Do not need to use all the invoking and excluding tests for a given $P$ and $F$

- $T = \{\text{a few carefully selected test cases}\}$
  - A few invoking tests (preferred to be the focused ones)
  - A few excluding tests

- Three notations
  - $\bigcup_{\text{invoking}} = \{\text{code executed by at least one invoking test in } T\}$
  - $\bigcup_{\text{excluding}} = \{\text{code executed by any tests in } T \text{ that do not exhibit } F\}$
  - $\bigcap_{\text{invoking}} = \{\text{code executed by every invoking test in } T\}$

---

**Heuristics for Finding Code Unique to a Feature (1)**

- Only one invoking test and one excluding test
  - Invoking test: the one with the **smallest** execution slice
  - Excluding test: the one with the **largest** execution slice

- $\bigcap_{\text{invoking}} - \bigcup_{\text{excluding}} = \{\text{code that is commonly executed by all invoking tests but not any excluding test}\}$

- Example
  - $F_\alpha$ can only be exhibited if either $F_\beta$ or $F_\gamma$ is also exhibited
    - All the invoking tests for $F_\alpha$ must also exhibit at least $F_\beta$ or $F_\gamma$ and perhaps many other features.
    - $F_\alpha$ has **no focused invoking tests**.
  - Use $\bigcap_{\text{invoking}} - \bigcup_{\text{excluding}}$ to find the code that is uniquely related to $F_\alpha$. 
Heuristics for Finding Code Unique to a Feature (2)

- $\cup_{\text{invoking}} - \cup_{\text{excluding}} = \{\text{code that is executed by any invoking test but not by any excluding test}\}

  - Example
  - $F_a$ is not bundled with $F_\beta$ or $F_\gamma$ in the way just described.
  - $F_a$ can be exhibited by itself without other features being exhibited simultaneously

Heuristics for Finding Code Common to Features

- Code common to $F_1$ and $F_2$ is what is executed by at least a test that exhibits only $F_1$ and not $F_2$, and at least a test that exhibits only $F_2$ and not $F_1$

  - $\cup_{\text{invoking}}$ for $F_1$ = $\{\text{code executed by tests which exhibit only } F_1 \text{ and no other features (or at least not } F_2)\}$

  - $\cup_{\text{invoking}}$ for $F_2$ = $\{\text{code executed by tests which exhibit only } F_2 \text{ and no other features (or at least not } F_1)\}$

  - Code common to $F_1$ and $F_2$ = $(\cup_{\text{invoking}}$ for $F_1) \cap (\cup_{\text{invoking}}$ for $F_2)$
Selecting Invoking and Excluding Tests (1)

- Different sets of code may be identified by different sets of invoking and excluding tests

- Poorly selected tests will lead to inaccurate identification
  - Example
    - Including code that is not unique to a given feature
    - Excluding code that should not be excluded

- Find code unique to a given feature
  - Invoking tests should be focused on the feature being located, if possible
  - Excluding tests should be as similar (in terms of execution slice) as possible to the invoking tests in order to filter out as much common code as possible

Selecting Invoking and Excluding Tests (2)

- Find code common to a group of features
  - For each feature, its invoking tests should be focused with respect to this feature, if possible
  - The invoking tests for a feature should be as dissimilar (in terms of execution slice) as possible to the invoking tests for other features in the group in order to exclude as much uncommon code as possible
Components

- Program components can be files, functions, blocks, decisions, c-uses, and p-uses

Requirements for Using \( \chi \) Vue

- \( \chi \) Vue is part of \( \chi \) Suds (a Software Understanding and Diagnosis System) developed at Telcordia (formerly Bellcore)
- Effective use of \( \chi \) Vue requires only that the programmer has a basic understanding of the program’s features and can identify some invoking as well as excluding tests
- By default, every test is in the dont_know category
Visualizing Features in Code Using \textit{Vue} (1)

- Slice 15 (counting \textit{characters} of the \textit{wordcount} program)
- Chapter 12 of the \textit{$\chi$Suds} User’s Manual

Visualizing Features in Code Using \textit{Vue} (2)
Visualizing Features in Code Using Vue (5)

```c
int main() {
    int len, width, charct;
    long lenvec = 0;
    long widthvec = 0;
    long charctvec = 0;
    int deltax = 0;
    int deword = 0;
    int docxer = 0;

    FILE *file;

    if (argc > 1 && argv[1][0] == '-') {
        for (p = argv[1]; *p++)
            switch(*p) {
            case 'd':
                deltax = 1;
                break;
            case 'w':
                deword = 1;
                break;
            case 'c':
                docxer = 1;
                break;
            default:
                printf(stderr, "Invalid option - %s", *p);
                return 1;
            }
        }
    }
    printf("%d %d %d\n", lenvec, widthvec, charctvec);
    return 0;
}
```

Vue

<table>
<thead>
<tr>
<th>File:</th>
<th>Line:</th>
<th>Coverage</th>
<th>Highlighting</th>
</tr>
</thead>
<tbody>
<tr>
<td>main.c</td>
<td>15 of 34</td>
<td>block</td>
<td>highest weight</td>
</tr>
</tbody>
</table>

Visualizing Features in Code Using Vue (6)

```c
int main() {
    int len, width, charct;
    long lenvec = 0;
    long widthvec = 0;
    long charctvec = 0;
    int deltax = 0;
    int deword = 0;
    int docxer = 0;

    FILE *file;

    if (argc > 1 && argv[1][0] == '-') {
        for (p = argv[1]; *p++)
            switch(*p) {
            case 'd':
                deltax = 1;
                break;
            case 'w':
                deword = 1;
                break;
            case 'c':
                docxer = 1;
                break;
            default:
                printf(stderr, "Invalid option - %s", *p);
                return 1;
            }
        }
    }
    printf("%d %d %d\n", lenvec, widthvec, charctvec);
    return 0;
}
```

Vue

<table>
<thead>
<tr>
<th>File:</th>
<th>Line:</th>
<th>Coverage</th>
<th>Highlighting</th>
</tr>
</thead>
<tbody>
<tr>
<td>main.c</td>
<td>16 of 34</td>
<td>block</td>
<td>highest weight</td>
</tr>
</tbody>
</table>
**A Case Study on SHARPE**

- **SHARPE**
  - A Symbolic Hierarchical Automated Reliability and Performance Evaluator
  - 35,412 lines of C code in 30 files
  - 373 functions
  - Examined five features are studied
    - MC (Markov Chain) ($f_1$)
    - MRM (Markov Reward Models) ($f_2$)
    - GSPN (Generalized Stochastic Petri-Nets) ($f_3$)
    - PFQN (Product-Form Queuing Networks) ($f_4$)
    - FT (Fault Trees) ($f_5$)

---

**Number of Blocks Unique to $F_{3,j}$ ($1 \leq j \leq 10$)**

<table>
<thead>
<tr>
<th></th>
<th>$F_1$</th>
<th>$F_2$</th>
<th>$F_3$</th>
<th>$F_4$</th>
<th>$F_5$</th>
<th>$F_{123}$</th>
<th>$F_{124}$</th>
<th>$F_{125}$</th>
<th>$F_{135}$</th>
<th>$F_{235}$</th>
<th>$F_{12345}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>$f_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_5$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- A blank entry means no block in the corresponding file is unique to the specified feature
- Only a small percentage of blocks are selected
### Number of Blocks Common to Each Pair of Features

<table>
<thead>
<tr>
<th>Feature 1</th>
<th>Feature 2</th>
<th>Blocks Common</th>
</tr>
</thead>
<tbody>
<tr>
<td>analysis</td>
<td>expectation</td>
<td>62 62 71 51 06 89 64 89 64 89</td>
</tr>
<tr>
<td>driver</td>
<td>machine</td>
<td>137 138 138 68 138 138 73 138 73 73</td>
</tr>
<tr>
<td>teacher</td>
<td>trainer</td>
<td>31 41 95 412 31 140 55 48 64</td>
</tr>
<tr>
<td>code</td>
<td>oracle</td>
<td>30 30 30</td>
</tr>
</tbody>
</table>

• The notation $F_1/F_2$ indicates code common to features $F_1$ and $F_2$

• A blank entry means no common block in the corresponding file

### Code Common to All Five Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Number of Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>analysis</td>
<td>62 62 71 51 06 89 64 89 64 89</td>
</tr>
<tr>
<td>machine</td>
<td>137 138 138 68 138 138 73 138 73 73</td>
</tr>
<tr>
<td>teacher</td>
<td>31 41 95 412 31 140 55 48 64</td>
</tr>
<tr>
<td>code</td>
<td>30 30 30</td>
</tr>
</tbody>
</table>

• A blank entry means no common code in the corresponding file
**Verification with SHARPE Experts**

- The identified *files, functions, blocks, and decisions* are either unique to a feature as they should be or shared by a pair of features or common to all five features.
- **No complete verification was done with respect to the identified c-uses and p-uses**
  - Very difficult for humans to have a complete understanding of a complicated system at such a fine granularity
  - *Some* identified c-uses and p-uses are verified and agreed on by the experts
- **Need more objective verification**
  - Ask experts to highlight code segments they think are important to each feature
  - Different segments might be highlighted by different people
  - Need to summarize such divergent information

---

**Conclusion (PART I)**

- Code identified using the execution slice-based technique (either unique to a feature or common to a group of features) can be used as a good starting point for studying program features
  - C-uses and p-uses provide an in-depth understanding
- This technique may *not* find all relevant code that makes up a feature
- Apply to the Y2K problems
  - Identify "*date-sensitive*" code which may be only a few lines in a system consisting of millions of lines of code
- **Extend to Program Debugging**
  - "*Invoking tests*" correspond to the "*failed tests*"
  - "*Excluding tests*" correspond to the "*successful tests*"
PART II

Quantifying the Disparity, Concentration, and Dedication between Program Components and Features


Click on the following pdf icon to download the paper

---

**Objective**

- Measure, *in a quantitative way*, the closeness between a feature and a program component
  - Example:
    - How much of the code in a program component is used to implement a given feature?
      - 50%, 70%, or more than 90%?
  - Previous studies highlight code that is *uniquely* related to a given feature.
    - Can be used as good starting points for locating program features
    - Does not identify where the *majority* of the code related to a given feature resides
  - Metrics proposed
    - Provide a much more complete picture of how a feature spreads over a software system
**Three Metrics**

- **Disparity**: measures how close a feature is to a program component
- **Concentration**: shows how much a feature is concentrated in a program component
- **Dedication**: indicates how much a program component is dedicated to a feature

**Represent a Feature Using a Set of Blocks**

- Represent an abstract feature in terms of some concrete program elements.
- Use the union of the execution slices of invoking inputs to find a set of code that is used to implement a given feature
  - Theoretically, we may need to use all the invoking inputs for a given program and feature
  - In practice, this is impossible and unnecessary
- An alternative is to use a good set of focused invoking inputs (which can be obtained from the regression test suite) to identify most of the code that is used to implement the feature
**Notation**

- Let $P$ be a program, $F$ a feature of $P$, $C$ a component of $P$, and $T$ a small set of carefully selected invoking inputs with the focus on $F$.
- $B_{C_{I}}$ is the set of blocks in $P$ executed by input $t_{I} \in T$.
- $B_{F}$ is the union of $B_{C_{I}}$ such that $t_{I} \in T$, i.e., the set of blocks in $P$ executed by at least one input in $T$. In other words, $B_{F}$ is a set of blocks in $P$ which are used to implement $F$.
- $B_{C}$ is the set of blocks in $C$.
- $B_{C} \cap F$ is the intersection of $B_{C}$ and $B_{F}$, i.e., the set of blocks in $C$ which are used to implement $F$.
- $B_{C} \cup F$ is the union of $B_{C}$ and $B_{F}$.
- $B_{C} \oplus F$ is the set of blocks in either $B_{C}$ or $B_{F}$, but not both, i.e., $B_{C} \oplus F$ equals $(B_{C} \cap B_{F}) \cup (B_{C} \cap \overline{B_{F}})$, where $\overline{B_{C}}$ and $\overline{B_{F}}$ are the complements of $B_{C}$ and $B_{F}$ in the set of blocks in $P$, respectively, $\overline{B_{C}} \cap B_{F}$ contains the blocks in $B_{F}$ but not in $B_{C}$, and $B_{C} \cap \overline{B_{F}}$ contains the blocks in $B_{C}$ but not in $B_{F}$.

**Disparity Metric: $\text{DISP}_{CF}$**

- $\text{DISP}_{CF} = \frac{|B_{C} \oplus F|}{|B_{C} \cup F|} = 1 - \frac{|B_{C} \cap F|}{|B_{C} \cup F|}$
- $0 \leq \text{DISP}_{CF} \leq 1$
  - Inversely proportional to the number of blocks in $B_{C} \cap F$
    - When $C$ and $F$ share more common blocks, their disparity should be smaller
  - Proportional to the number of blocks in $B$
    - The more blocks in either $B_{C}$ or $B_{F}$, but not both, the larger the disparity between $C$ and $F$
- $\text{DISP}_{CF} = 1$ if and only if there is no common block between $B_{C}$ and $B_{F}$
  - $B_{C} \cap F = \emptyset$
- $\text{DISP}_{CF} = 0$ if and only if feature $F$ is totally implemented in component $C$ and every block in $C$ is used to implement $F$
  - $B_{C} \cap F = B_{C} \cup F$ (i.e., $B_{C} = B_{F}$)
**Concentration Metric: CONC_{FC}**

- \( \text{CONC}_{FC} = \frac{|B_C \cap F|}{|B_F|} \)
- \( 0 \leq \text{CONC}_{FC} \leq 1 \)
- Inversely proportional to the number of blocks in \( B_F \)
  - When \( B_F \) has more blocks, it is less likely for all the blocks to reside in the same component
- Proportional to the number of blocks in \( B_{C \cap F} \)
  - When \( C \) and \( F \) have more blocks in common, they have a bigger commitment to each other
- \( \text{CONC}_{FC} = 1 \) if and only if all the blocks used to implement \( F \) are in \( C \)
- \( \text{CONC}_{FC} = 0 \) if and only if none of these blocks is in \( C \)

**Dedication Metric: DEDI_{CF}**

- \( \text{DEDI}_{CF} = \frac{|B_C \cap F|}{|B_C|} \)
- \( 0 \leq \text{DEDI}_{CF} \leq 1 \)
- Inversely proportional to the number of blocks in \( B_C \)
  - When \( B_C \) has more blocks, it is more likely that some of these blocks have nothing to do with \( F \)
- Proportional to the number of blocks in \( B_{C \cap F} \)
  - When \( C \) and \( F \) have more blocks in common, they have a bigger commitment to each other
- \( \text{DEDI}_{CF} = 1 \) if and only if all the blocks in \( C \) are used to implement \( F \)
- \( \text{DEDI}_{CF} = 0 \) if and only if none of these blocks has anything to do with \( F \)
Possible Relationship between $B_F$ and $B_C$

- Case I: $B_F \cap B_C = \emptyset$
  
  $DISP_{CF} = 1$, $CONC_{FC}$ = $DEDI_{CF} = 0$

- Case II: $B_C$ and $B_F$ have some blocks in common
  (i.e., $B_C \cap B_F \neq \emptyset$, but neither includes the other):
  $0 < DISP_{CF}$, $CONC_{FC}$ and $DEDI_{CF} < 1$

- Case III: $B_C$ equals $B_F$
  (i.e., $B_C \cap B_F = B_C$):
  $DISP_{CF} = 0$, $CONC_{FC}$ = $DEDI_{CF} = 1$

- Case IV: $B_F$ is a subset of $B_C$ and $B_F \neq B_C$
  (i.e., $B_F \subset B_C$):
  $0 < DISP_{CF}$ and $DEDI_{CF} < 1$, $CONC_{FC} = 1$

- Case V: $B_C$ is a subset of $B_F$ and $B_C \neq B_F$
  (i.e., $B_C \subset B_F$):
  $0 < DISP_{CF}$ and $CONC_{FC} < 1$, $DEDI_{CF} = 1$

**DISP_{CF}, CONC_{FC} and DEDI_{CF}**

- A 100% concentration or a 100% dedication does not guarantee a zero disparity between $F$ and $C$.

- Disparity is 0 if and only if both concentration and dedication are 100%, (i.e., $B_F = B_C$)

- Disparity is 1 if and only if both concentration and dedication are 0, (i.e., $B_C \cap B_F = \emptyset$)

- If $CONC_{FC} \neq 0$ (i.e., $B_C \cap B_F \neq \emptyset$) then $DEDI_{CF} \neq 0$, and vice versa

- If $CONC_{FC} = 0$ (i.e., $B_C \cap B_F = \emptyset$) then $DEDI_{CF} = 0$, and vice versa
**Example (1)**

- 2 components $C_1$ and $C_2$
- 2 features $F_1$ and $F_2$
- 2 invoking inputs ($t_1$ and $t_2$) focused on $F_1$

**Assume**

\[
B_{F_1} = B_{t_1} \cup B_{t_2} = \{h_{11}, h_{12}, h_{13}, h_{14}\} \cup \{h_{15}, h_{16}, h_{17}, h_{18}\} = \{h_{11}, h_{12}, h_{13}, h_{14}, h_{15}, h_{16}, h_{17}, h_{18}\}
\]

**We have**

\[
\begin{align*}
B_{C_1 \cap F_1} &= \{h_{11}, h_{12}, h_{13}, h_{14}\} \\
B_{C_2 \cap F_1} &= \{h_{15}, h_{16}, h_{17}, h_{18}\} \\
B_{C_1 \cup F_1} &= \{h_{11}, h_{12}, h_{13}, h_{14}, h_{15}, h_{16}, h_{17}, h_{18}\} \\
B_{C_2 \cup F_1} &= \{h_{15}, h_{16}, h_{17}, h_{18}\} \\
B_{C_1 \oplus F_1} &= \{h_{11}, h_{12}, h_{13}, h_{14}\} \\
B_{C_2 \oplus F_1} &= \{h_{15}, h_{16}, h_{17}, h_{18}\}
\end{align*}
\]

Each cell in the diagram is a block. The pink ones are related to the feature $F_1$.

**Example (2)**

- The disparity

\[
\begin{align*}
DISP_{C_1,F_1} &= 1 - \frac{|B_{C_1 \cap F_1}|}{|B_{C_1 \cup F_1}|} = 1 - \frac{4}{9} = 0.556 \\
DISP_{C_2,F_1} &= 1 - \frac{|B_{C_2 \cap F_1}|}{|B_{C_2 \cup F_1}|} = 1 - \frac{4}{10} = 0.60
\end{align*}
\]

- The concentration

\[
\begin{align*}
CONC_{F_1,C_1} &= \frac{|B_{C_1 \cap F_1}|}{|B_{F_1}|} = \frac{4}{8} = 0.50 \\
CONC_{F_1,C_2} &= \frac{|B_{C_2 \cap F_1}|}{|B_{F_1}|} = \frac{4}{8} = 0.50
\end{align*}
\]

- The dedication

\[
\begin{align*}
DEDIC_{F_1,C_1} &= \frac{|B_{C_1 \cap F_1}|}{|B_{C_1}|} = \frac{4}{8} = 0.80 \\
DEDIC_{F_1,C_2} &= \frac{|B_{C_2 \cap F_1}|}{|B_{C_2}|} = \frac{4}{8} = 0.667
\end{align*}
\]
A Case Study on SHARPE

- First developed in 1986
- 35,412 lines of C code in 30 files
- 373 functions and 11752 blocks

<table>
<thead>
<tr>
<th>File</th>
<th># of blocks</th>
<th>File</th>
<th># of blocks</th>
<th>File</th>
<th># of blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>analyse.c</td>
<td>334</td>
<td>indent.c</td>
<td>240</td>
<td>pfgn.c</td>
<td>226</td>
</tr>
<tr>
<td>bind.c</td>
<td>911</td>
<td>initfile.c</td>
<td>604</td>
<td>phase.c</td>
<td>544</td>
</tr>
<tr>
<td>bilike.c</td>
<td>75</td>
<td>logmode.c</td>
<td>506</td>
<td>readgraph.c</td>
<td>524</td>
</tr>
<tr>
<td>cexpo.c</td>
<td>606</td>
<td>makefile.c</td>
<td>176</td>
<td>read1.c</td>
<td>421</td>
</tr>
<tr>
<td>cpg.c</td>
<td>202</td>
<td>mphpg.c</td>
<td>229</td>
<td>results.c</td>
<td>564</td>
</tr>
<tr>
<td>debug.c</td>
<td>64</td>
<td>mita.c</td>
<td>134</td>
<td>share.c</td>
<td>762</td>
</tr>
<tr>
<td>expe.c</td>
<td>186</td>
<td>multpath.c</td>
<td>128</td>
<td>noe.c</td>
<td>269</td>
</tr>
<tr>
<td>ftree.c</td>
<td>903</td>
<td>newcpg.c</td>
<td>230</td>
<td>symbol.c</td>
<td>527</td>
</tr>
<tr>
<td>insize.c</td>
<td>511</td>
<td>newlinese.c</td>
<td>604</td>
<td>uniform.c</td>
<td>227</td>
</tr>
<tr>
<td>insize.c</td>
<td>604</td>
<td>newpharse.c</td>
<td>414</td>
<td>util.c</td>
<td>407</td>
</tr>
</tbody>
</table>

- Six features
  – MC (Markov Chain)
  – FT (Fault Trees)
  – GSPN (Generalized Stochastic Petri-Nets)
  – PFQN (Product-Form Queueing Networks)
  – RELG (Reliability Graphs)
  – MRM (Markov Reward Models)

Data Collection

- Each c file as a separate program component
- A set of invoking inputs focused on each feature was carefully selected from the regression test suite of SHARPE
  - Advantages of using regression tests
    - They were real inputs used during the integration and system testing
    - There exist clear descriptions for many of these tests, which made it very easy to select invoking inputs focused on a given feature

- Execution slice in terms of blocks of each of these inputs was computed
- Computed also were $B_F$ for each of the six features and $B_C$ for each of the 30 files of SHARPE
- Using $B_F$’s and $B_C$’s, compute $DEDI_{CF}$, $CONC_{FC}$ and $DISP_{CF}$ with respect to every possible pair of $F$ and $C$
**Data Verification**

- Need an oracle
  - Design documentation
    - Well-designed system: a clear mapping between each feature and its corresponding code segments
    - Experts who have a good knowledge of the system being analyzed
      - May not exist
      - Time consuming & not affordable
      - How to summarize divergent information
  - Present each $B_F$ to experts who are familiar with SHARPE
    - The identified blocks are used to implement the designated feature as they should be
    - No additional block that has to be added to each $B_F$

---

**Feature Concentration in Components**

<table>
<thead>
<tr>
<th>File</th>
<th>CC</th>
<th>PF</th>
<th>OCPH</th>
<th>PPQM</th>
<th>REGO</th>
<th>MMUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>analysis.c</td>
<td>0.0105</td>
<td>0.0109</td>
<td>0.0094</td>
<td>0.0090</td>
<td>0.0109</td>
<td></td>
</tr>
<tr>
<td>blith.c</td>
<td>0.0124</td>
<td>0.0124</td>
<td>0.0124</td>
<td>0.0124</td>
<td>0.0124</td>
<td></td>
</tr>
<tr>
<td>crypto.c</td>
<td>0.0134</td>
<td>0.0134</td>
<td>0.0134</td>
<td>0.0134</td>
<td>0.0134</td>
<td></td>
</tr>
<tr>
<td>cg.c</td>
<td>0.0145</td>
<td>0.0145</td>
<td>0.0145</td>
<td>0.0145</td>
<td>0.0145</td>
<td></td>
</tr>
<tr>
<td>debug.c</td>
<td>0.0156</td>
<td>0.0156</td>
<td>0.0156</td>
<td>0.0156</td>
<td>0.0156</td>
<td></td>
</tr>
<tr>
<td>free.c</td>
<td>0.0168</td>
<td>0.0168</td>
<td>0.0168</td>
<td>0.0168</td>
<td>0.0168</td>
<td></td>
</tr>
<tr>
<td>ln.cpp.c</td>
<td>0.0179</td>
<td>0.0179</td>
<td>0.0179</td>
<td>0.0179</td>
<td>0.0179</td>
<td></td>
</tr>
<tr>
<td>incbase.c</td>
<td>0.0191</td>
<td>0.0191</td>
<td>0.0191</td>
<td>0.0191</td>
<td>0.0191</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0201</td>
<td>0.0201</td>
<td>0.0201</td>
<td>0.0201</td>
<td>0.0201</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0212</td>
<td>0.0212</td>
<td>0.0212</td>
<td>0.0212</td>
<td>0.0212</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0223</td>
<td>0.0223</td>
<td>0.0223</td>
<td>0.0223</td>
<td>0.0223</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0234</td>
<td>0.0234</td>
<td>0.0234</td>
<td>0.0234</td>
<td>0.0234</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0245</td>
<td>0.0245</td>
<td>0.0245</td>
<td>0.0245</td>
<td>0.0245</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0256</td>
<td>0.0256</td>
<td>0.0256</td>
<td>0.0256</td>
<td>0.0256</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0267</td>
<td>0.0267</td>
<td>0.0267</td>
<td>0.0267</td>
<td>0.0267</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0278</td>
<td>0.0278</td>
<td>0.0278</td>
<td>0.0278</td>
<td>0.0278</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0289</td>
<td>0.0289</td>
<td>0.0289</td>
<td>0.0289</td>
<td>0.0289</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0300</td>
<td>0.0300</td>
<td>0.0300</td>
<td>0.0300</td>
<td>0.0300</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0311</td>
<td>0.0311</td>
<td>0.0311</td>
<td>0.0311</td>
<td>0.0311</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0322</td>
<td>0.0322</td>
<td>0.0322</td>
<td>0.0322</td>
<td>0.0322</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0333</td>
<td>0.0333</td>
<td>0.0333</td>
<td>0.0333</td>
<td>0.0333</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0344</td>
<td>0.0344</td>
<td>0.0344</td>
<td>0.0344</td>
<td>0.0344</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0355</td>
<td>0.0355</td>
<td>0.0355</td>
<td>0.0355</td>
<td>0.0355</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0366</td>
<td>0.0366</td>
<td>0.0366</td>
<td>0.0366</td>
<td>0.0366</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0377</td>
<td>0.0377</td>
<td>0.0377</td>
<td>0.0377</td>
<td>0.0377</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0388</td>
<td>0.0388</td>
<td>0.0388</td>
<td>0.0388</td>
<td>0.0388</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0399</td>
<td>0.0399</td>
<td>0.0399</td>
<td>0.0399</td>
<td>0.0399</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0410</td>
<td>0.0410</td>
<td>0.0410</td>
<td>0.0410</td>
<td>0.0410</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0421</td>
<td>0.0421</td>
<td>0.0421</td>
<td>0.0421</td>
<td>0.0421</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0432</td>
<td>0.0432</td>
<td>0.0432</td>
<td>0.0432</td>
<td>0.0432</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0443</td>
<td>0.0443</td>
<td>0.0443</td>
<td>0.0443</td>
<td>0.0443</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0454</td>
<td>0.0454</td>
<td>0.0454</td>
<td>0.0454</td>
<td>0.0454</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0465</td>
<td>0.0465</td>
<td>0.0465</td>
<td>0.0465</td>
<td>0.0465</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0476</td>
<td>0.0476</td>
<td>0.0476</td>
<td>0.0476</td>
<td>0.0476</td>
<td></td>
</tr>
<tr>
<td>inslib.c</td>
<td>0.0487</td>
<td>0.0487</td>
<td>0.0487</td>
<td>0.0487</td>
<td>0.0487</td>
<td></td>
</tr>
</tbody>
</table>

**Sum** | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00

Blank entry means the corresponding feature concentration is zero.
## Component Dedication to Features

<table>
<thead>
<tr>
<th>PID</th>
<th>MC</th>
<th>PE</th>
<th>GSPF</th>
<th>FPQH</th>
<th>RELQ</th>
<th>MRM</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>analyte</td>
<td>0.1376</td>
<td>0.0748</td>
<td>0.4433</td>
<td>0.2485</td>
<td>0.0599</td>
<td>0.5627</td>
<td>1.5699</td>
</tr>
<tr>
<td>knot</td>
<td>0.1315</td>
<td>0.1299</td>
<td>0.4568</td>
<td>0.2688</td>
<td>0.1246</td>
<td>0.3834</td>
<td>0.7966</td>
</tr>
<tr>
<td>boost</td>
<td>0.4133</td>
<td>0.4039</td>
<td>0.5946</td>
<td>0.4028</td>
<td>0.3067</td>
<td>0.7611</td>
<td>1.8021</td>
</tr>
<tr>
<td>cpc</td>
<td>0.4109</td>
<td>0.3796</td>
<td>0.7376</td>
<td>0.7277</td>
<td>0.1905</td>
<td>0.8227</td>
<td>1.8792</td>
</tr>
<tr>
<td>dbi</td>
<td>0.4104</td>
<td>0.4080</td>
<td>0.4381</td>
<td>0.1994</td>
<td>0.4081</td>
<td>0.5827</td>
<td>1.2022</td>
</tr>
<tr>
<td>en</td>
<td>0.1515</td>
<td>0.0678</td>
<td>0.2536</td>
<td>0.3200</td>
<td>0.2817</td>
<td>0.1344</td>
<td>1.8495</td>
</tr>
<tr>
<td>en-gm</td>
<td>0.4104</td>
<td>0.4123</td>
<td>0.1723</td>
<td>1.2022</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>in</td>
<td>0.2385</td>
<td>0.2532</td>
<td>0.2910</td>
<td>0.2668</td>
<td>0.2795</td>
<td>0.3445</td>
<td>1.7957</td>
</tr>
<tr>
<td>lm</td>
<td>0.4104</td>
<td>0.4094</td>
<td>0.2901</td>
<td>0.2951</td>
<td>0.2411</td>
<td>0.6066</td>
<td>0.9449</td>
</tr>
<tr>
<td>ml</td>
<td>0.1468</td>
<td>0.1578</td>
<td>0.4518</td>
<td>0.3585</td>
<td>0.5478</td>
<td>0.0478</td>
<td>0.0478</td>
</tr>
<tr>
<td>meta</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.1304</td>
<td>1.1304</td>
</tr>
<tr>
<td>multi</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.2038</td>
<td>1.2038</td>
</tr>
<tr>
<td>multi</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.7266</td>
<td>1.7266</td>
</tr>
<tr>
<td>multi</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2.0925</td>
<td>2.0925</td>
</tr>
<tr>
<td>multi</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2.2552</td>
<td>2.2552</td>
</tr>
<tr>
<td>multi</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2.7156</td>
<td>2.7156</td>
</tr>
<tr>
<td>multi</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2.8089</td>
<td>2.8089</td>
</tr>
<tr>
<td>multi</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2.9309</td>
<td>2.9309</td>
</tr>
<tr>
<td>multi</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2.9874</td>
<td>2.9874</td>
</tr>
</tbody>
</table>

Blank entry means the corresponding component dedication is zero.

## Disparity between Features and Components

<table>
<thead>
<tr>
<th>PID</th>
<th>MC</th>
<th>PE</th>
<th>GSPF</th>
<th>FPQH</th>
<th>RELQ</th>
<th>MRM</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>analyte</td>
<td>0.1376</td>
<td>0.0748</td>
<td>0.4433</td>
<td>0.2485</td>
<td>0.0599</td>
<td>0.5627</td>
<td>1.5699</td>
</tr>
<tr>
<td>knot</td>
<td>0.1315</td>
<td>0.1299</td>
<td>0.4568</td>
<td>0.2688</td>
<td>0.1246</td>
<td>0.3834</td>
<td>0.7966</td>
</tr>
<tr>
<td>boost</td>
<td>0.4133</td>
<td>0.4039</td>
<td>0.5946</td>
<td>0.4028</td>
<td>0.3067</td>
<td>0.7611</td>
<td>1.8021</td>
</tr>
<tr>
<td>cpc</td>
<td>0.4109</td>
<td>0.3796</td>
<td>0.7376</td>
<td>0.7277</td>
<td>0.1905</td>
<td>0.8227</td>
<td>1.8792</td>
</tr>
<tr>
<td>dbi</td>
<td>0.4104</td>
<td>0.4080</td>
<td>0.4381</td>
<td>0.1994</td>
<td>0.4081</td>
<td>0.5827</td>
<td>1.2022</td>
</tr>
<tr>
<td>en</td>
<td>0.1515</td>
<td>0.0678</td>
<td>0.2536</td>
<td>0.3200</td>
<td>0.2817</td>
<td>0.1344</td>
<td>1.8495</td>
</tr>
<tr>
<td>en-gm</td>
<td>0.4104</td>
<td>0.4123</td>
<td>0.1723</td>
<td>1.2022</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>in</td>
<td>0.2385</td>
<td>0.2532</td>
<td>0.2910</td>
<td>0.2668</td>
<td>0.2795</td>
<td>0.3445</td>
<td>1.7957</td>
</tr>
<tr>
<td>lm</td>
<td>0.4104</td>
<td>0.4094</td>
<td>0.2901</td>
<td>0.2951</td>
<td>0.2411</td>
<td>0.6066</td>
<td>0.9449</td>
</tr>
<tr>
<td>ml</td>
<td>0.1468</td>
<td>0.1578</td>
<td>0.4518</td>
<td>0.3585</td>
<td>0.5478</td>
<td>0.0478</td>
<td>0.0478</td>
</tr>
<tr>
<td>meta</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.1304</td>
<td>1.1304</td>
</tr>
<tr>
<td>multi</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.2038</td>
<td>1.2038</td>
</tr>
<tr>
<td>multi</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.7266</td>
<td>1.7266</td>
</tr>
<tr>
<td>multi</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2.0925</td>
<td>2.0925</td>
</tr>
<tr>
<td>multi</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2.2552</td>
<td>2.2552</td>
</tr>
<tr>
<td>multi</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2.7156</td>
<td>2.7156</td>
</tr>
<tr>
<td>multi</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2.8089</td>
<td>2.8089</td>
</tr>
<tr>
<td>multi</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2.9309</td>
<td>2.9309</td>
</tr>
<tr>
<td>multi</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>2.9874</td>
<td>2.9874</td>
</tr>
</tbody>
</table>

Blank entry means the corresponding component dedication is zero.
Conclusion (PART II)(1)

- SHARPE has a very delocalized structure with features spread over many files
  - For a given feature, in general, it has less than 8% concentration in a file
  - For a given file, if it is used to implement a feature, it normally has at least 20% of its blocks dedicated to this feature
  - Why?
    - Features are incrementally incorporated into SHARPE, rather than planned in the original design

Conclusion (PART II)(2)

- Three metrics are proposed (disparity, concentration and dedication) to provide a good quantitative measure of the closeness between a feature and a program component
- Help programmers capture more precisely where each feature resides in the system
  - A quantitative measure computed based on carefully defined metrics versus a qualitative understanding obtained from intuitive feeling
- Our metrics are complementary to other program comprehension techniques to help software programmers better understand the system at hand
PART III

*Measuring Distance between Program Features*


Click on the following pdf icon to download the paper

---

**Objective**

- Develop a metric to determine the distance between features
  - How are features of a system close to each other *in a quantitative way*?
    - Is the distance between features $\alpha$ and $\beta$ larger than that between $\beta$ and $\gamma$?
    - If so, how much larger?
  - Provide a good start to understanding how a modification made to one feature is likely to affect other features
Our Approach

- Represent an **abstract feature** in terms of **some concrete program** elements.
  - Use an execution slice-based technique to identify a set of code (basic blocks in our case) that is used to implement each feature
    - A basic block (also known as a block) is a sequence of consecutive code containing no branch such that if part of the code is executed, other code will also be executed
  - An input is an **invoking input** with respect to a feature if, when executed on the program, it shows the functionality of that feature.

- Compute distance between two features
  - **Static distance**
    - in terms of code that is used to **implement these features**
  - **Dynamic distance**
    - in terms of code that is **executed by inputs which exhibit these features**

Notation

- $P$ is a program; $\alpha$, $\beta$ and $\gamma$ are features of $P$.
- $T_{\alpha}$ is a small set of carefully selected invoking inputs with the focus on $\alpha$.
- $B_{\alpha}$ is the set of blocks in $P$ executed by input $t_{\alpha} \in T_{\alpha}$. Depending on whether the execution count of each block is considered, an appropriate weight factor may have to be assigned to it.
- $B_{\alpha}$ which equals the union of $B_{\alpha}$ for all $t_{\alpha} \in T_{\alpha}$ is the set of blocks in $P$ which are used to implement $\alpha$. A similar definition applies to $B_{\beta}$ and $B_{\gamma}$.
- $B_{\alpha} \cap \beta$ is the set of blocks shared by features $\alpha$ and $\beta$.
- $B_{\alpha} \cup \beta$ is the set of blocks in the union of $B_{\alpha}$ and $B_{\beta}$ (i.e., $B_{\alpha} \cup B_{\beta}$).
- $B_{\alpha \oplus \beta}$ is the set of blocks in either $B_{\alpha}$ or $B_{\beta}$, but not both, i.e., $B_{\alpha \oplus \beta}$ equals $(B_{\alpha} \cap B_{\beta}) \cup (B_{\alpha} \cap B_{\beta})$, where $B_{\alpha}$ and $B_{\beta}$ are the complements of $B_{\alpha}$ and $B_{\beta}$ in the set of blocks in $P$, respectively. $B_{\alpha} \cap B_{\beta}$ contains the blocks in $B_{\beta}$ but not in $B_{\alpha}$, and $B_{\alpha} \cap B_{\beta}$ contains the blocks in $B_{\alpha}$ but not in $B_{\beta}$.
- $DIST_{\alpha \beta}$ is the distance between features $\alpha$ and $\beta$. 
**Properties**

- The numerical value of $DIST_{\alpha\beta}$ must be normalized between 0 and 1 (i.e., $0 \leq DIST_{\alpha\beta} \leq 1$) so that the distance between two features can be compared in a meaningful way.
- The value assigned should be a monotonically decreasing function of the number of blocks in $B_{\alpha\beta}$, i.e., the more blocks in the intersection of $B_{\alpha}$ and $B_{\beta}$, the smaller the distance between $\alpha$ and $\beta$. This makes sense because when $\alpha$ and $\beta$ share more common blocks, their distance should be smaller.
- The value assigned should be a monotonically increasing function of the number of blocks in $B_{\alpha\beta}$, i.e., the more blocks in either $B_{\alpha}$ or $B_{\beta}$, but not both, the larger the distance between $\alpha$ and $\beta$. This implies that when there are more blocks in $B_{\alpha}$ but not in $B_{\beta}$, or vice versa, the distance between these two should also be larger.
- The value 1 should be assigned if and only if there is no common block between $B_{\alpha}$ and $B_{\beta}$, i.e., the intersection between $B_{\alpha}$ and $B_{\beta}$ is empty.
- The value 0 should be assigned if and only if features $\alpha$ and $\beta$ use exactly the same set of blocks, i.e., every block in $B_{\alpha}$ is in $B_{\beta}$, and vice versa.

$^a$Notation $\Phi$ represents the exclusive or relation between two sets.

**Distance Metric**

$$DIST'_{\alpha\beta} = \frac{|B_{\alpha} \Phi B_{\beta}|}{|B_{\alpha} \cup B_{\beta}|}$$

This leads to the computation:

$$= \frac{|B_{\alpha} + |B_{\beta}| - 2 \cdot |B_{\alpha} \cap B_{\beta}|}{|B_{\alpha}| + |B_{\beta}| - |B_{\alpha} \cap B_{\beta}|}$$

$$= 1 - \frac{|B_{\alpha} \cap B_{\beta}|}{|B_{\alpha}| + |B_{\beta}| - |B_{\alpha} \cap B_{\beta}|}$$

$$= 1 - \frac{|B_{\alpha} \setminus B_{\beta}|}{|B_{\alpha}|}$$

where $|B_{\alpha}|$ represents the number of elements in set $B_{\alpha}$, and so on.
### Three Axioms

- $\text{DIST}_{\alpha \alpha} = 0$
- $\text{DIST}_{\alpha \beta} = \text{DIST}_{\beta \alpha}$
- $\text{DIST}_{\alpha \beta} + \text{DIST}_{\beta \gamma} \geq \text{DIST}_{\alpha \gamma}$

### Observation

![Diagram showing possible relationships between $B_\alpha$ and $B_\beta$.](image)

- **Case I:** $B_\alpha \cap B_\beta = \emptyset$. In this case, $\text{DIST}_{\alpha \beta} = 1$.
- **Case II:** $B_\alpha$ and $B_\beta$ have some blocks in common, i.e., $B_\alpha \cap B_\beta \neq \emptyset$, but neither ensharpen the other. Here, $\text{DIST}_{\alpha \beta}$ is between 0 and 1.
- **Case III:** $B_\alpha$ equals $B_\beta$ which makes $B_\alpha \cap B_\beta = B_\alpha$. As a result, $\text{DIST}_{\alpha \beta} = 0$.
- **Case IV:** $B_\alpha$ is a subset of $B_\beta$ (i.e., $B_\alpha \subseteq B_\beta$) but $B_\beta \neq B_\alpha$. $\text{DIST}_{\alpha \beta}$ is between 0 and 1.
- **Case V:** $B_\alpha$ is a subset of $B_\beta$ (i.e., $B_\alpha \subseteq B_\beta$) but $B_\beta \neq B_\alpha$. $\text{DIST}_{\alpha \beta}$ is also between 0 and 1.
Static Distance versus Dynamic Distance (1)

- Depending on whether the execution frequency of each block is considered during the construction of the sets of code

- **Static distance**
  - Only depends on how features are implemented in the system
  - The execution frequency of each block is not used in computing the distance

- **Dynamic distance**
  - Depends on how each feature is implemented
  - Also takes into account how each feature is executed based on a user's operational profile

- Static distance gives the closeness of two features from the system implementation point of view, whereas the dynamic distance presents such closeness from a user's execution point of view

Static Distance versus Dynamic Distance (2)

- Static distance between two features is fixed once their implementation is completed, but the dynamic distance changes depending on how these features are executed

- The dynamic distance computed using one user's operational profile can be different from that using another profile even though the corresponding static distance stays the same
**Static Distance versus Dynamic Distance (3)**

• If the static distance between two features is unity (i.e., no overlap in the implementation), the dynamic distance between these two features must also be unity

\[
\text{static distance} = 1 \implies \text{dynamic distance} = 1 \text{ (YES)}
\]

- *Is the reverse true?*
  - Consider "error handling code" used for two features but not executed
    - Dynamic distance equals one but not the static distance
    - In general, do NOT expect this to happen

\[
\text{dynamic distance} = 1 \implies \text{static distance} = 1 \text{ (not necessarily)}
\]

**Static Distance versus Dynamic Distance (4)**

• A zero static distance between two features (i.e., the same set of code is used to implement both features), does not necessarily give a zero dynamic distance
  - Different blocks may have different execution counts

\[
\text{static distance} = 0 \implies \text{dynamic distance} = 0 \text{ (not necessarily)}
\]

• A zero dynamic distance between two features does not imply a zero static distance
  - Different error handling routines are implemented for these two features.
    The dynamic distance can be zero if none of their invoking inputs trigger the errors.
  - Nevertheless, after the error routines are taken into account, the static distance is greater than zero.

\[
\text{dynamic distance} = 0 \implies \text{static distance} = 0 \text{ (not necessarily)}
\]
Static Distance versus Dynamic Distance (5)

- For two given features, if their static distance is between zero and one, their dynamic distance, in general, is also between zero and one, and vice versa.
- A smaller static distance does not imply a smaller dynamic distance, and vice versa.
- A larger static distance does not imply a larger dynamic distance, and vice versa.

A Case Study on SHARPE (1)

- A Symbolic Hierarchical Automated Reliability and Performance Evaluator
  - 35,412 lines of C code in 30 files
  - 373 functions
  - Examined 5 features
    - Fault Trees (FT) ($f_1$)
    - Markov Chains (MC) ($f_2$)
    - Generalized Stochastic Petri-Nets (GSPN) ($f_3$)
    - Product-Form Queuing Networks (PFQN) ($f_4$)
    - Reliability Graph (RELG) ($f_5$)
### Static Distance between Each Pair of Features in SHARPE

<table>
<thead>
<tr>
<th>File</th>
<th>$d_1$</th>
<th>$d_2$</th>
<th>$d_3$</th>
<th>$d_4$</th>
<th>$d_5$</th>
<th>$d_6$</th>
<th>$d_7$</th>
<th>$d_8$</th>
<th>$d_9$</th>
<th>$d_{10}$</th>
<th>$d_{11}$</th>
<th>$d_{12}$</th>
<th>$d_{13}$</th>
<th>$d_{14}$</th>
<th>$d_{15}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>analyte</td>
<td>1</td>
<td>1</td>
<td>0.256</td>
<td>0.300</td>
<td>0.361</td>
<td>1</td>
<td>1</td>
<td>0.256</td>
<td>0.300</td>
<td>0.361</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>blot</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.168</td>
<td>0.184</td>
<td>0.184</td>
<td>1</td>
<td>1</td>
<td>0.168</td>
<td>0.184</td>
<td>0.184</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>first91</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ensemble</td>
<td>0.286</td>
<td>0.364</td>
<td>0.361</td>
<td>0.331</td>
<td>0.533</td>
<td>0.495</td>
<td>1</td>
<td>1</td>
<td>0.533</td>
<td>0.495</td>
<td>0.331</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>elfg</td>
<td>0.160</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>elfgc</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>feature</td>
<td>1</td>
<td>1</td>
<td>0.368</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>featureinnen</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>file.c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>featureinnen.c</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>featureinneninnen</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>featureinneninneninnen</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>featureinneninneninneninnen</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>featureinneninneninneninneninnen</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$^a$Notation $d_1 d_2$ represents the static distance between $d_1$ and $d_2$ $(D^S, D^P)$, and so on.

### Static Distance between Each Pair of Features in SHARPE

<table>
<thead>
<tr>
<th>File</th>
<th>$d_1$</th>
<th>$d_2$</th>
<th>$d_3$</th>
<th>$d_4$</th>
<th>$d_5$</th>
<th>$d_6$</th>
<th>$d_7$</th>
<th>$d_8$</th>
<th>$d_9$</th>
<th>$d_{10}$</th>
<th>$d_{11}$</th>
<th>$d_{12}$</th>
<th>$d_{13}$</th>
<th>$d_{14}$</th>
<th>$d_{15}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>analyte</td>
<td>1</td>
<td>1</td>
<td>0.256</td>
<td>0.300</td>
<td>0.361</td>
<td>1</td>
<td>1</td>
<td>0.256</td>
<td>0.300</td>
<td>0.361</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>blot</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.168</td>
<td>0.184</td>
<td>0.184</td>
<td>1</td>
<td>1</td>
<td>0.168</td>
<td>0.184</td>
<td>0.184</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>first91</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ensemble</td>
<td>0.286</td>
<td>0.364</td>
<td>0.361</td>
<td>0.331</td>
<td>0.533</td>
<td>0.495</td>
<td>1</td>
<td>1</td>
<td>0.533</td>
<td>0.495</td>
<td>0.331</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>elfg</td>
<td>0.160</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>elfgc</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>feature</td>
<td>1</td>
<td>1</td>
<td>0.368</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>featureinnen</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>file.c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>featureinnen.c</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>featureinneninnen</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>featureinneninneninnen</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>featureinneninneninneninnen</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>featureinneninneninneninneninnen</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>featureinneninneninneninneninneninnen</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$^a$Notation $d_1 d_2$ represents the dynamic distance between $d_1$ and $d_2$ $(D^S, D^P)$, and so on.
A Case Study on SHARPE (2)

- SHARPE has a very delocalized structure with features spread over many files
  - Confirmed by those who are very familiar with SHARPE
    - Features are incrementally incorporated into SHARPE, rather than planned in the original design
- Provide quantitative measurements (both static and dynamic) to help programmers accurately capture how far two features are from each other rather than rely on some intuitive feel for the system which provides only a qualitative description of whether two features are close to each other.

Conclusion (PART III)

- Our metrics can be used to measure the distance between two features
- The distance measurement can serve as a good starting point to understanding how a modification made to one feature is likely to affect other features
  - \( \text{DIST}_{\alpha \beta} < \text{DIST}_{\alpha \gamma} \) modifications to \( \alpha \) can very possibly have a higher impact on \( \beta \) than on \( \gamma \)
- Allow programmers to understand the possibility for interactions between features from a different perspective
- Our next step is to investigate how our metrics complement other methodologies for better detecting of interactions between program features
Overall Conclusion

• A set of techniques are developed to help programmers *understand their code in a more cost effective way*