# Visualization of Test Information to Assist Fault Localization

James A. Jones, Mary Jean Harrold, John Stasko

ICSE '02 , May 19-25, 2002, Orlando, Florida, USA.

# About the Authors

James A. Jones

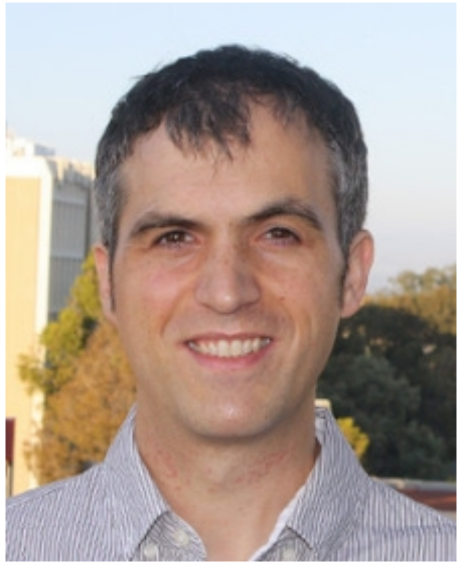
Mary Jean Harrold

John Stasko

# About the Authors

Alessandro Orso

James A. Jones

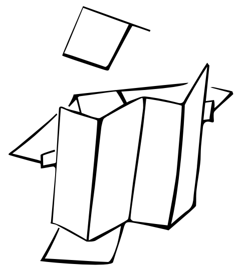

Francisco Servant

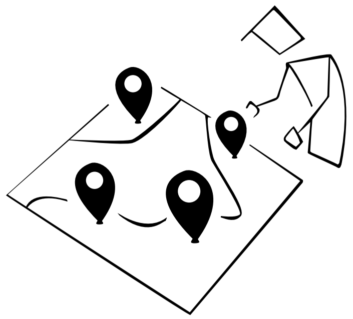
## About the Paper

- 2015 ACM SIGSOFT Impact Award.
  - Research that has had extraordinary
  - Granted to only one research paper per year
  - Paper must be published at least ten years prior.

## Outline

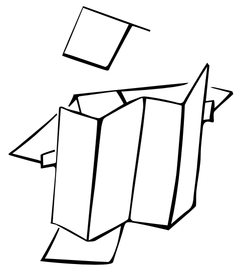
- Problem statement
- Background knowledge
- Method
- Evaluation
- Related work
- Future work
- Conclusion
- Discussion questions

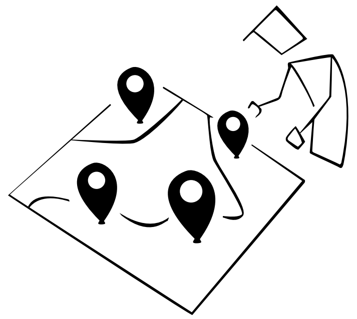




# Outline

- **Problem statement**
- Background knowledge
- Method
- Evaluation
- Related work
- Future work
- Conclusion
- Discussion questions



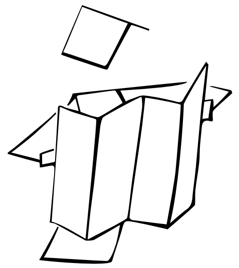


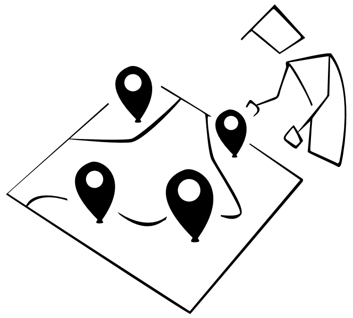
## Problem Statement

- Software debugging
  - Locating errors is the most difficult component of debugging tasks
- Fault localization
  - Reducing the number of delivered faults
  - Estimated to consume 50% to 80% of the development and maintenance effort

## Outline

- Problem statement
- **Background knowledge**
- Method
- Evaluation
- Related work
- Future work
- Conclusion
- Discussion questions



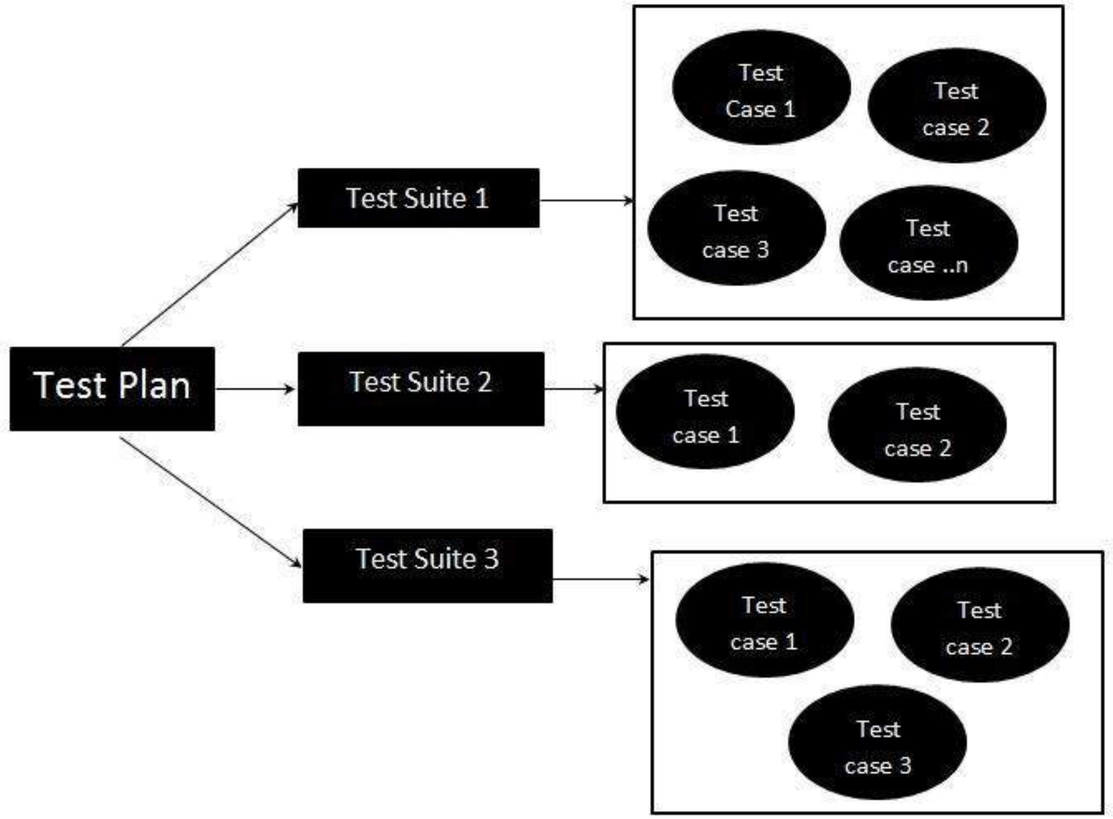


## Background Knowledge

- Debugging process
  - (1) identify statements involved in failures
  - **(2) narrow the search**
    - by selecting suspicious statements that might contain faults
  - (3) hypothesize about suspicious faults
  - (4) restore program variables to a specific state

# Background Knowledge

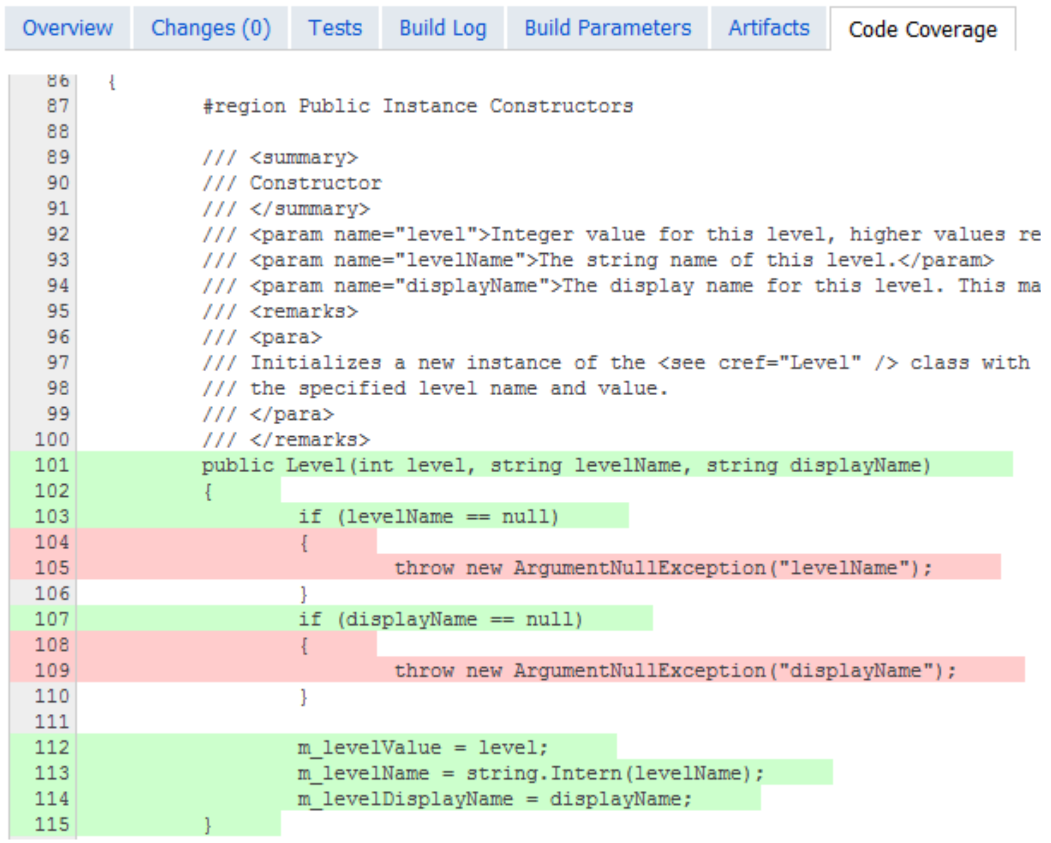
- Test suite
  - Collection of test cases
    - test a software program
    - show the program has some specified set of behaviors



https://www.tutorialspoint.com/software_testing_dictionary/test_suite.htm

# Background Knowledge

- Code coverage
  - Percentage of code covered by automated tests
  - Statements in a body of code executed through a test run

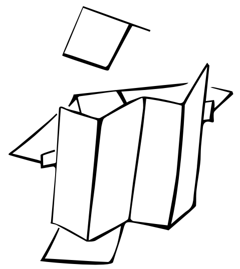


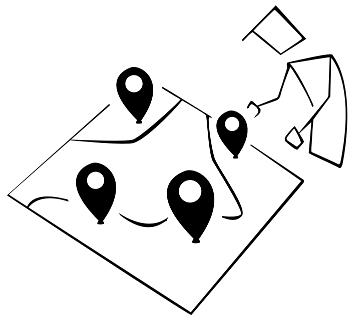
```
{
        #region Public Instance Constructors

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="level">Integer value for this level, higher values re
        /// <param name="levelName">The string name of this level.</param>
        /// <param name="displayName">The display name for this level. This ma
        /// <remarks>
        /// <para>
        /// Initializes a new instance of the <see cref="Level" /> class with
        /// the specified level name and value.
        /// </para>
        /// </remarks>
        public Level(int level, string levelName, string displayName)
        {
                if (levelName == null)
                {
                        throw new ArgumentNullException("levelName");
                }
                if (displayName == null)
                {
                        throw new ArgumentNullException("displayName");
                }

                m_levelValue = level;
                m_levelName = string.Intern(levelName);
                m_levelDisplayName = displayName;
        }
```

https://www.jetbrains.com/dotcover/features/

## Outline

- Problem statement
- Background knowledge
- **Method**
- Evaluation
- Related work
- Future work
- Conclusion
- Discussion questions



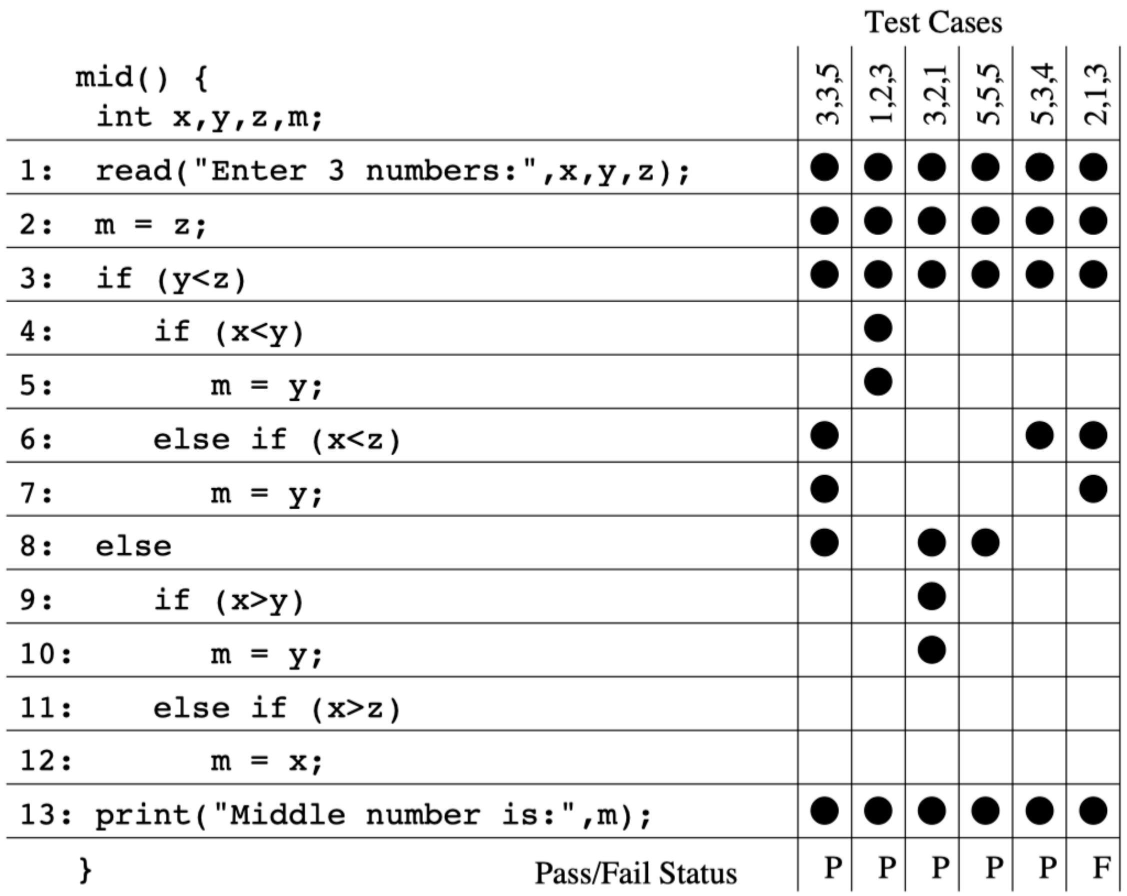


## Method

- Visualization technique
  - Provide global view of the test suite execution
  - Visually map the participation of each program statement
    - visual mapping of passed and failed test cases
    - identification of potential faulty statements.

# Method

- Technique input
  - Source code
  - Pass/fail results
  - Code coverage

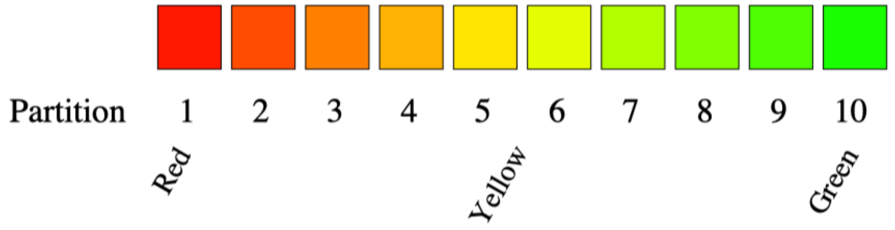


|  |  | Test Cases |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
|  | `mid() {` `int x,y,z,m;` | 3,3,5 | 1,2,3 | 3,2,1 | 5,5,5 | 5,3,4 | 2,1,3 |
| 1: | `read("Enter 3 numbers:",x,y,z);` | ● | ● | ● | ● | ● | ● |
| 2: | `m = z;` | ● | ● | ● | ● | ● | ● |
| 3: | `if (y<z)` | ● | ● | ● | ● | ● | ● |
| 4: | `if (x<y)` |  | ● |  |  |  |  |
| 5: | `m = y;` |  | ● |  |  |  |  |
| 6: | `else if (x<z)` | ● |  |  |  | ● | ● |
| 7: | `m = y;` | ● |  |  |  |  | ● |
| 8: | `else` | ● |  | ● | ● |  |  |
| 9: | `if (x>y)` |  |  | ● |  |  |  |
| 10: | `m = y;` |  |  | ● |  |  |  |
| 11: | `else if (x>z)` |  |  |  |  |  |  |
| 12: | `m = x;` |  |  |  |  |  |  |
| 13: | `print("Middle number is:",m);` | ● | ● | ● | ● | ● | ● |
|  | `}` Pass/Fail Status | P | P | P | P | P | F |

## Method

- Coloring technique
  - Color Component

$$\text{color(s)} = \text{low color (red)} + \frac{\%\text{passed(s)}}{\%\text{passed(s)} + \%\text{failed(s)}} * \text{color range}$$

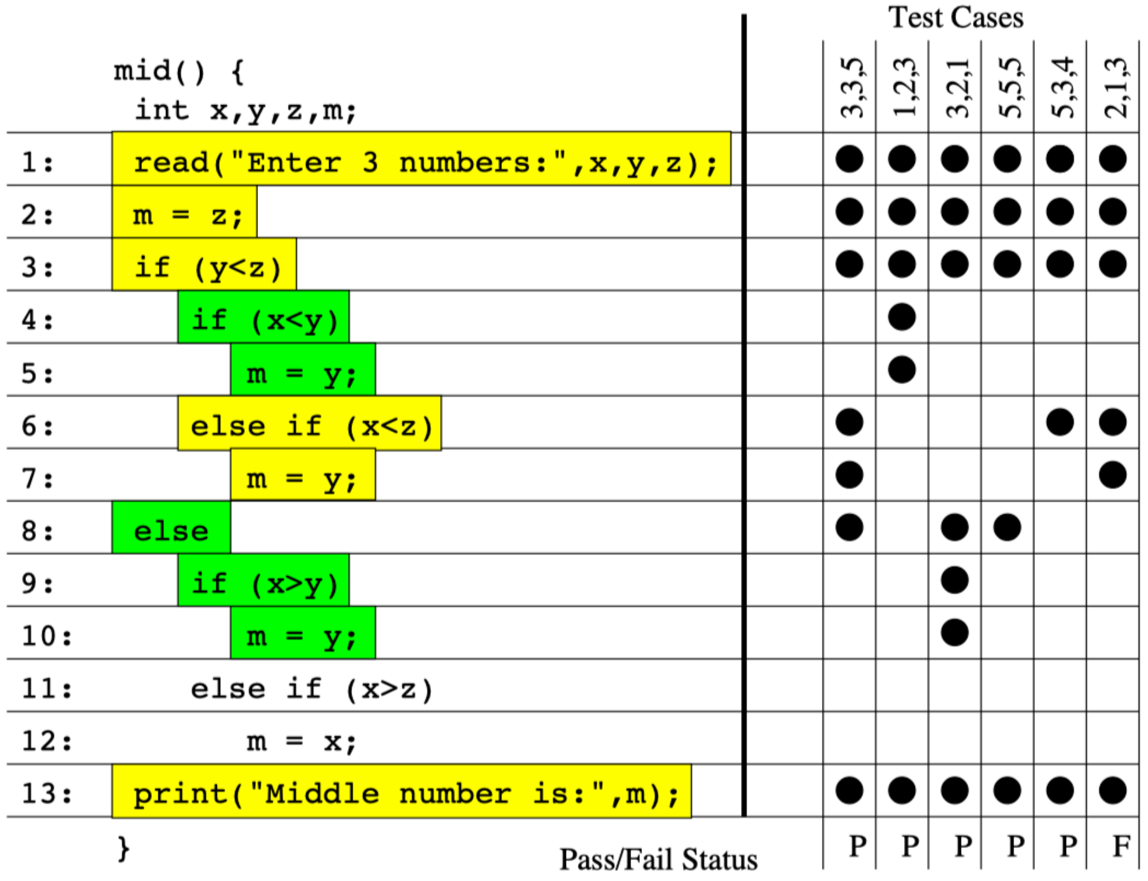


  - Brightness Component

$$\text{bright(s)} = \max(\%\ \text{passed(s)}, \%\ \text{failed(s)})$$

## Method

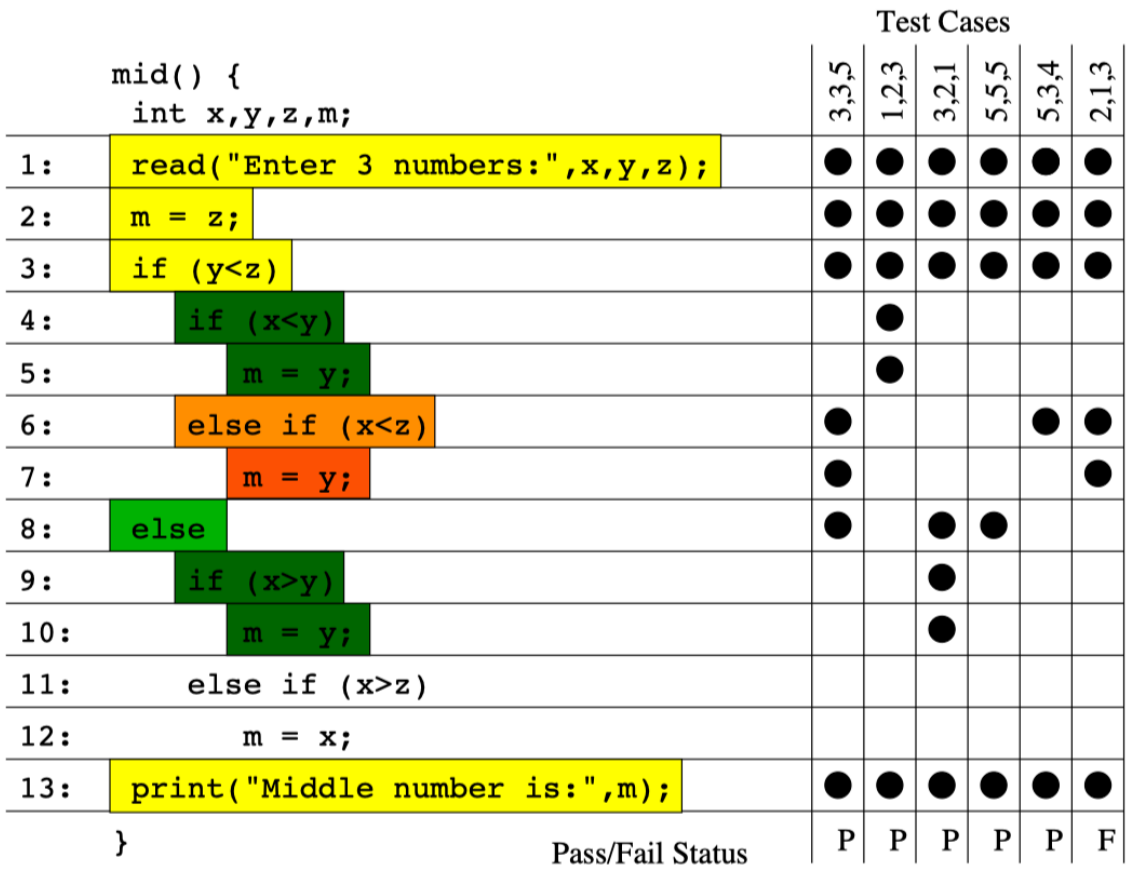
- Coloring technique
  - Color Component



|  |  | Test Cases |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
|  | mid() { |  |  |  |  |  |  |
|  | int x,y,z,m; | 3,3,5 | 1,2,3 | 3,2,1 | 5,5,5 | 5,3,4 | 2,1,3 |
| 1: | 🟨 `read("Enter 3 numbers:",x,y,z);` | ● | ● | ● | ● | ● | ● |
| 2: | 🟨 `m = z;` | ● | ● | ● | ● | ● | ● |
| 3: | 🟨 `if (y<z)` | ● | ● | ● | ● | ● | ● |
| 4: | 🟩 `    if (x<y)` |  | ● |  |  |  |  |
| 5: | 🟩 `        m = y;` |  | ● |  |  |  |  |
| 6: | 🟨 `    else if (x<z)` | ● |  |  |  | ● | ● |
| 7: | 🟨 `        m = y;` | ● |  |  |  |  | ● |
| 8: | 🟩 `else` | ● |  | ● | ● |  |  |
| 9: | 🟩 `    if (x>y)` |  |  | ● |  |  |  |
| 10: | 🟩 `        m = y;` |  |  | ● |  |  |  |
| 11: | `    else if (x>z)` |  |  |  |  |  |  |
| 12: | `        m = x;` |  |  |  |  |  |  |
| 13: | 🟨 `print("Middle number is:",m);` | ● | ● | ● | ● | ● | ● |
|  | } Pass/Fail Status | P | P | P | P | P | F |

# Method

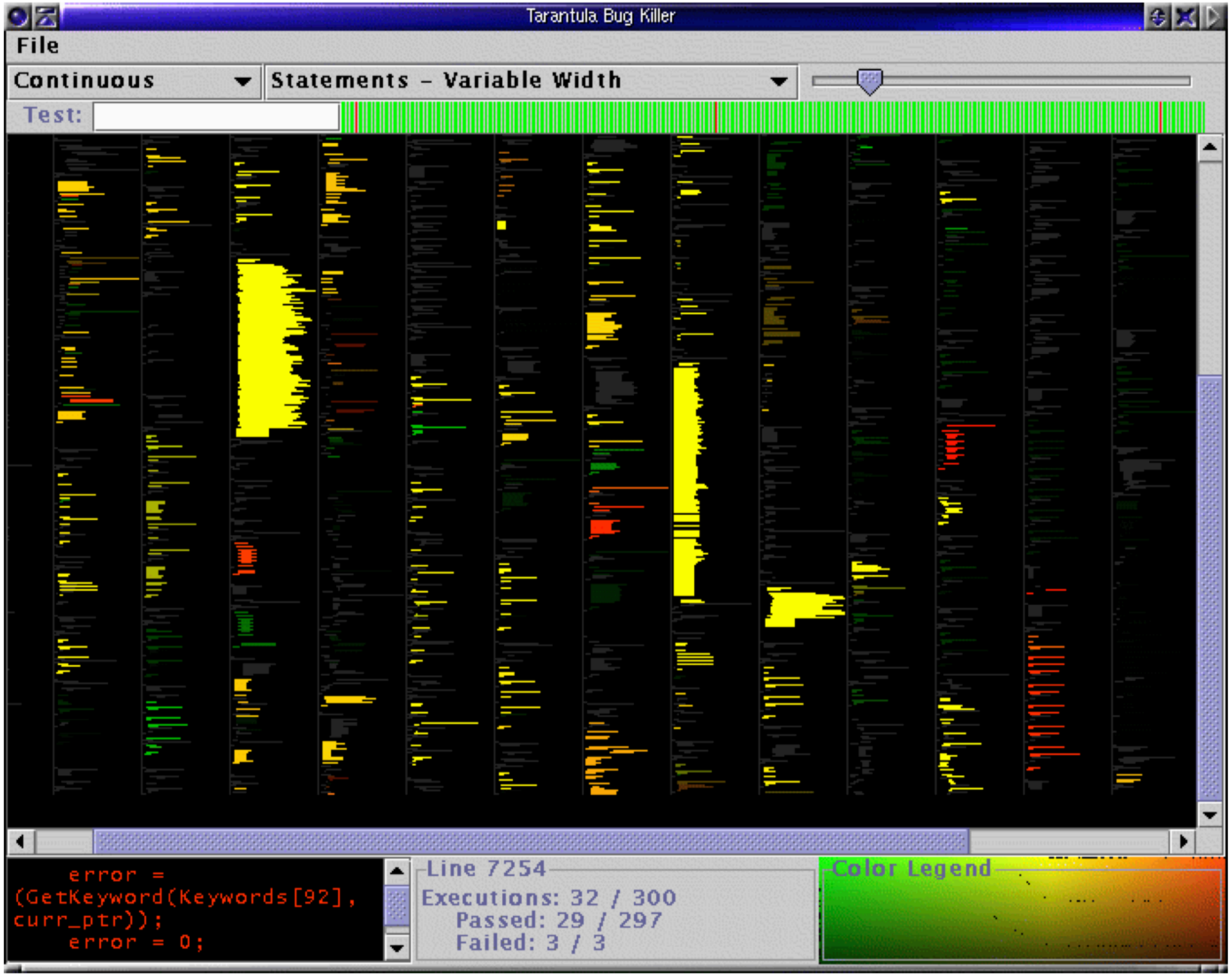
- Coloring technique
  - Color Component
  - Brightness Component



| | | Test Cases | | | | | |
|---|---|---|---|---|---|---|---|
| | `mid() {` | | | | | | |
| | `int x,y,z,m;` | 3,3,5 | 1,2,3 | 3,2,1 | 5,5,5 | 5,3,4 | 2,1,3 |
| 1: | `read("Enter 3 numbers:",x,y,z);` | ● | ● | ● | ● | ● | ● |
| 2: | `m = z;` | ● | ● | ● | ● | ● | ● |
| 3: | `if (y<z)` | ● | ● | ● | ● | ● | ● |
| 4: | `    if (x<y)` | | ● | | | | |
| 5: | `        m = y;` | | ● | | | | |
| 6: | `    else if (x<z)` | ● | | | | ● | ● |
| 7: | `        m = y;` | ● | | | | | ● |
| 8: | `else` | ● | | ● | ● | | |
| 9: | `    if (x>y)` | | | ● | | | |
| 10: | `        m = y;` | | | ● | | | |
| 11: | `    else if (x>z)` | | | | | | |
| 12: | `        m = x;` | | | | | | |
| 13: | `print("Middle number is:",m);` | ● | ● | ● | ● | ● | ● |
| | `}` Pass/Fail Status | P | P | P | P | P | F |

## Method

- Prototype (Tarantula)

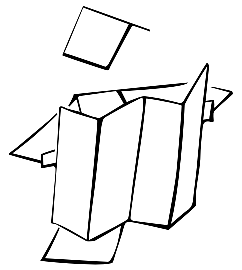
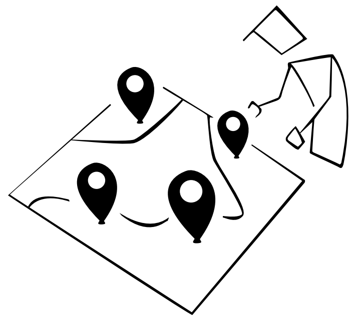


```
    error =
(GetKeyword(Keywords[92],
curr_ptr));
    error = 0;
```

Line 7254
Executions: 32 / 300
Passed: 29 / 297
Failed: 3 / 3

# Outline

- Problem statement
- Background knowledge
- Method
- **Evaluation**
- Related work
- Future work
- Conclusion
- Discussion questions

# Evaluation

- Effectiveness of the technique illumination of faulty statements
- Subject program (Space)
  - Language: C
  - 9564 lines
  - 33 associated versions
    - single fault each
- Test pool for Space
  - 10,000 test cases
    - instrumented the program for coverage
      - 30 test cases that exercised nearly every statement and edge
    - final test pool of 13,585 test cases

## Evaluation

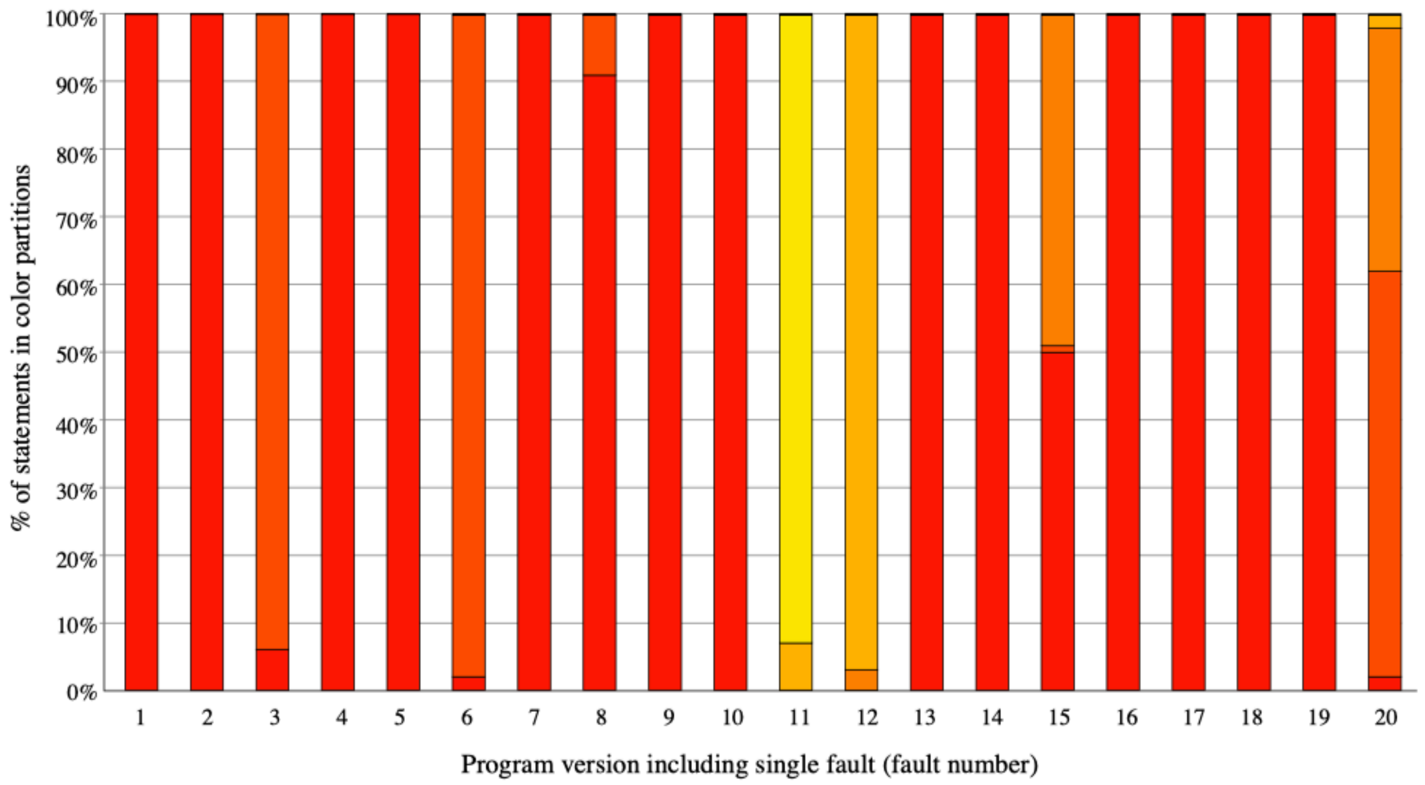
- Selected evaluation tests
  - 1000 randomly sized
  - Randomly generated
- Evaluation questions
  - How often does our technique color the faulty statement(s) in a program red or in a reddish color?
    - False negatives - technique fails to color the faulty statements red
  - How often does our technique color nonfaulty statements in a program red or in a reddish color?
    - False positives - technique colors nonfaulty statements red.

# Evaluation

- Evaluation studies
  - Study 1: Single-fault Versions
    - Evaluate against a program with one fault
  - Study 2: Multiple-fault Versions
    - Evaluate against a program with 2, 3, 4, and 5 faults

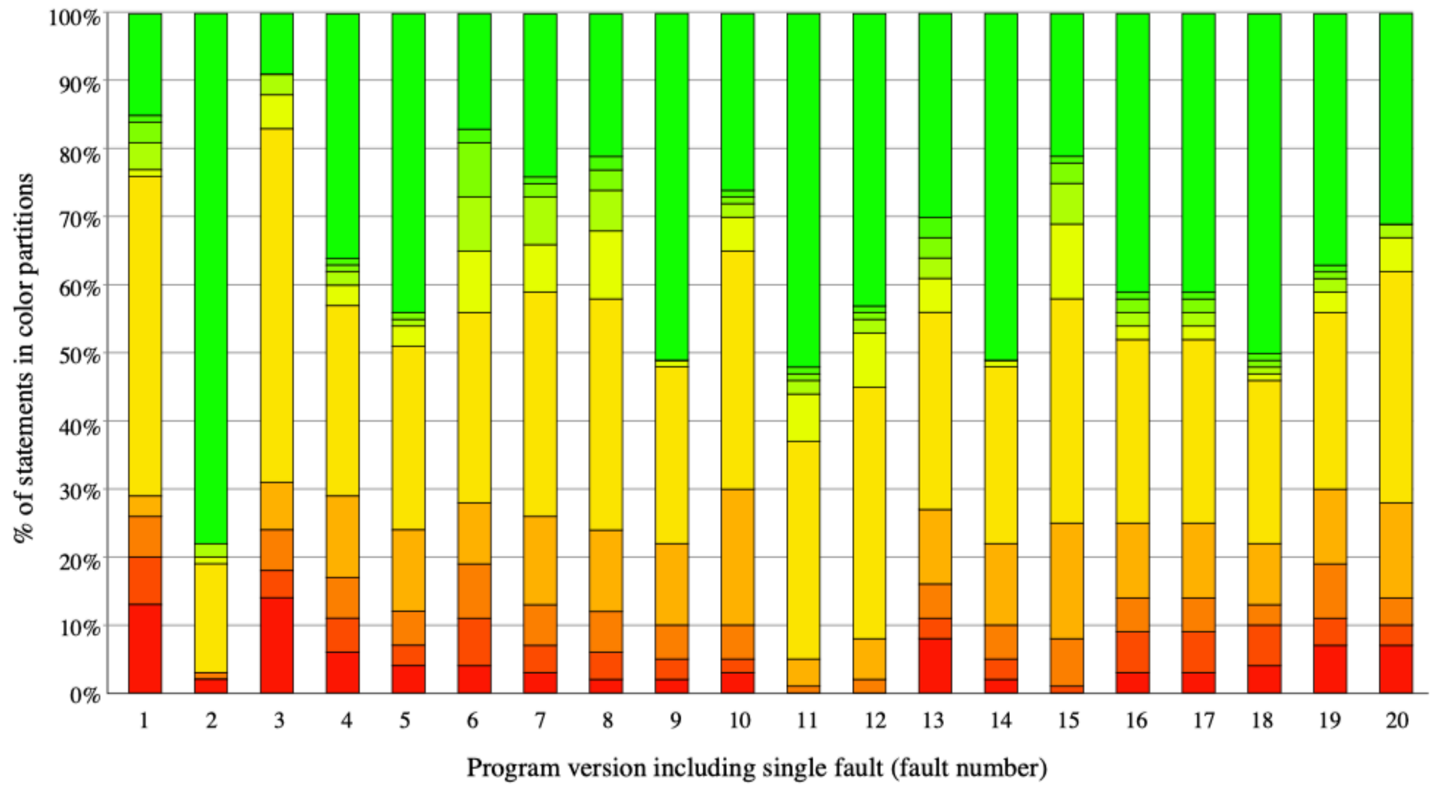
# Evaluation

- Study 1: Single-fault Versions



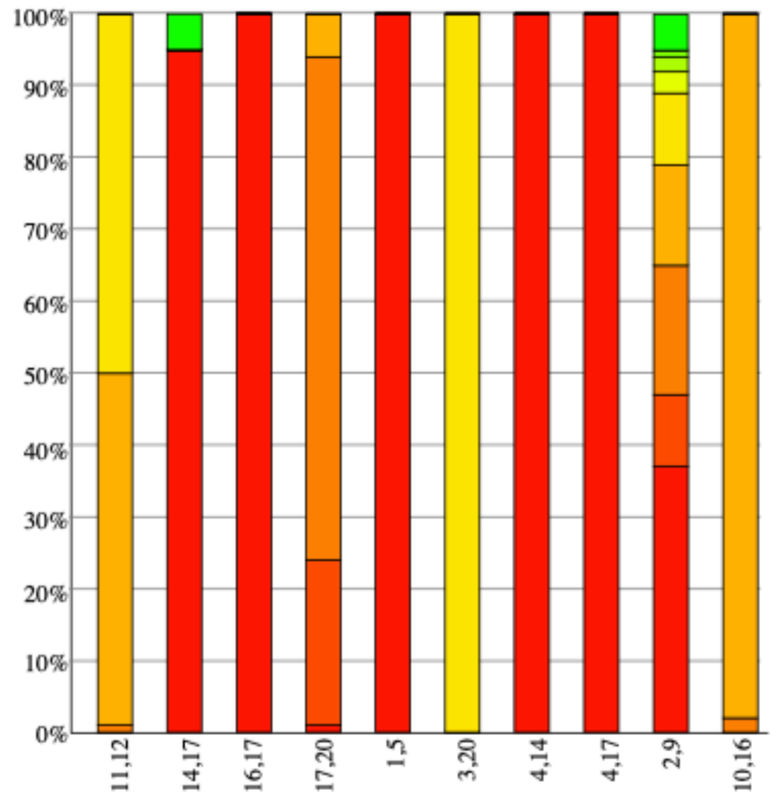
# Evaluation

- Study 1: Single-fault Versions

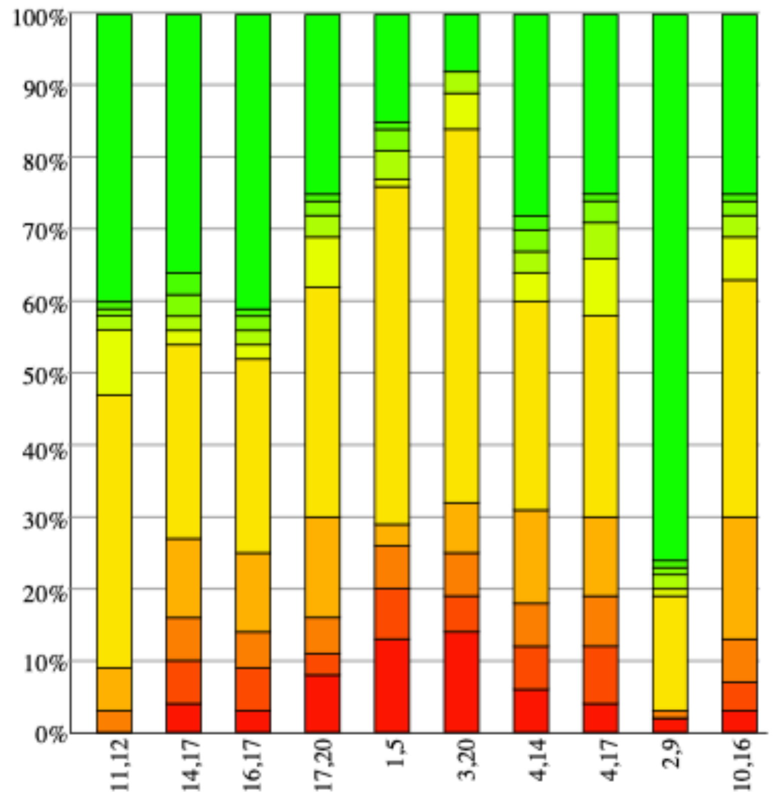


# Evaluation

- Study 2: Multiple-fault Versions

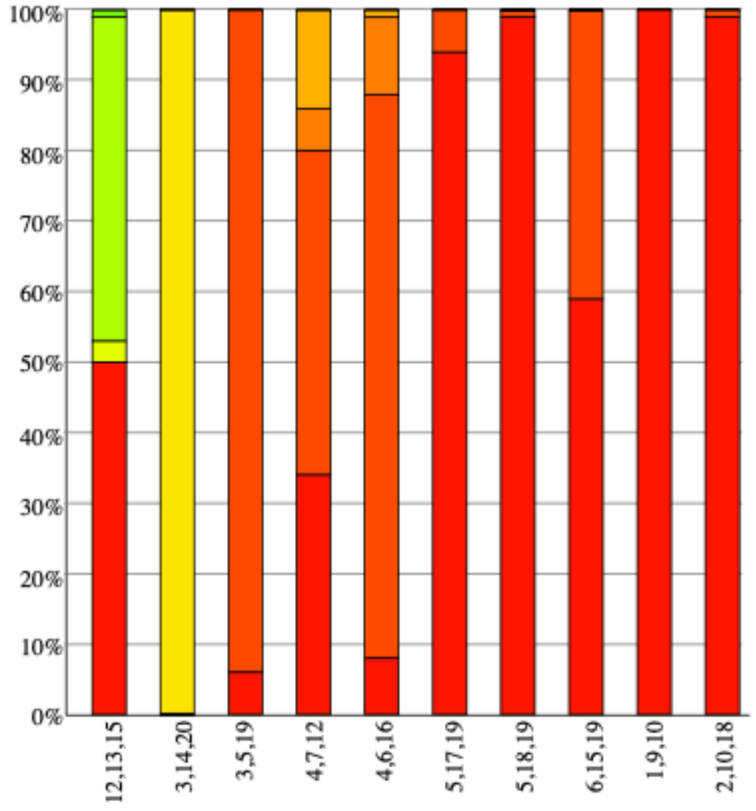


2–fault versions

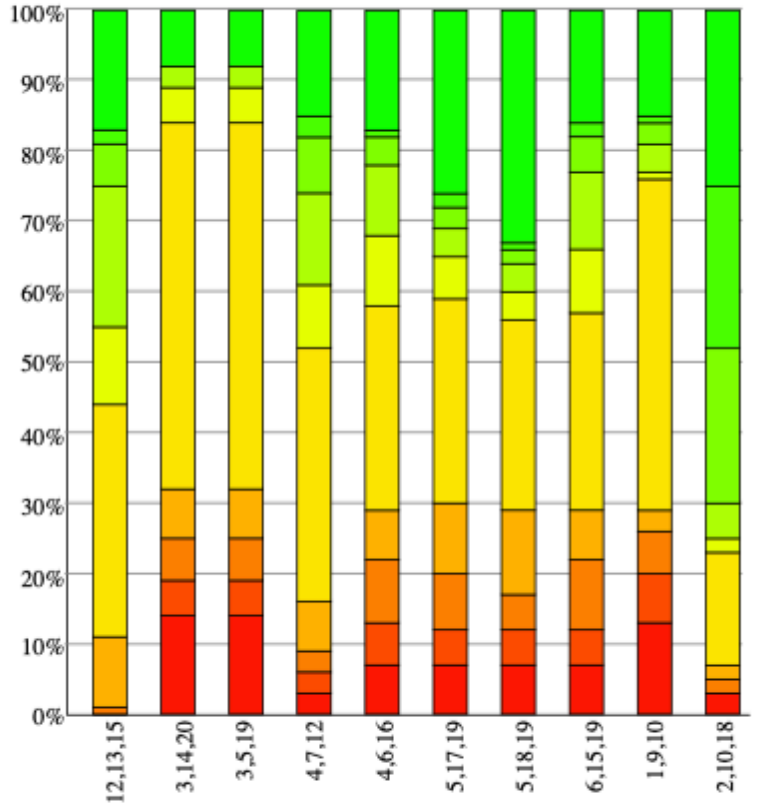


# Evaluation

- Study 2: Multiple-fault Versions

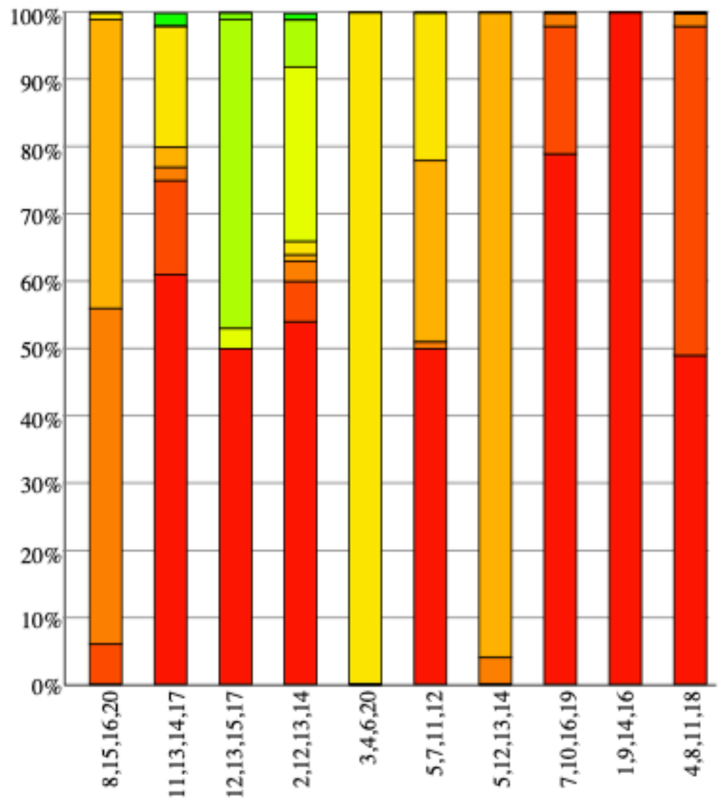


3–fault versions

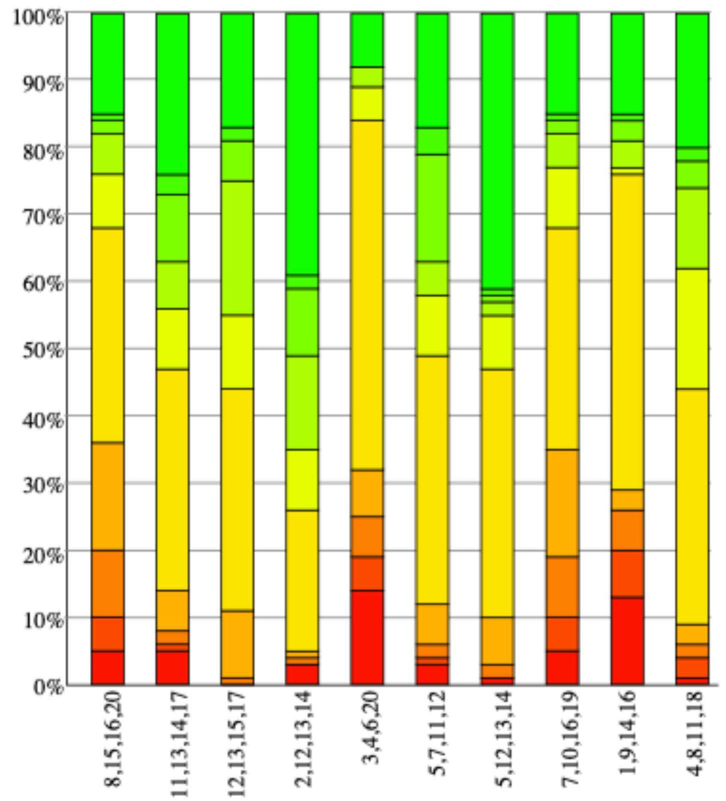


# Evaluation

- Study 2: Multiple-fault Versions

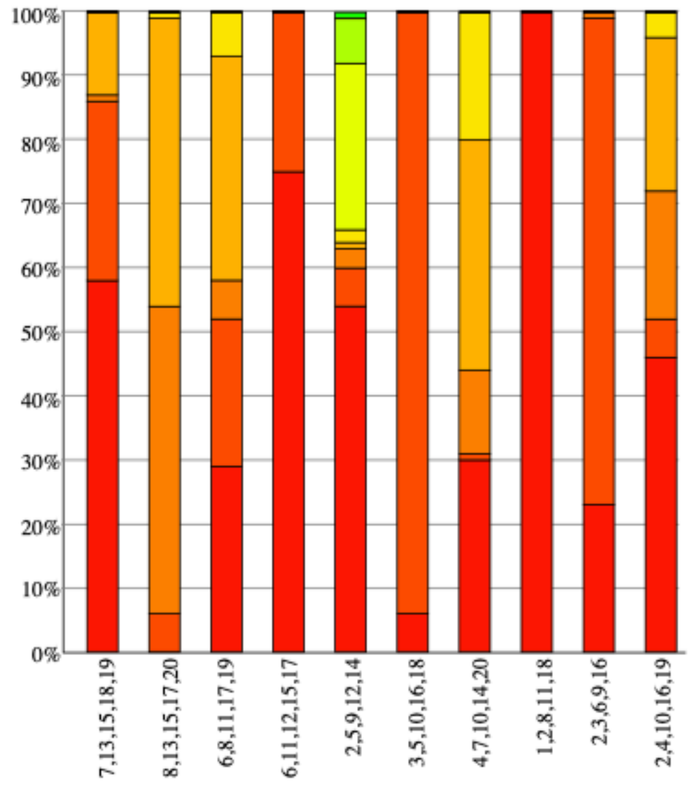


4–fault versions

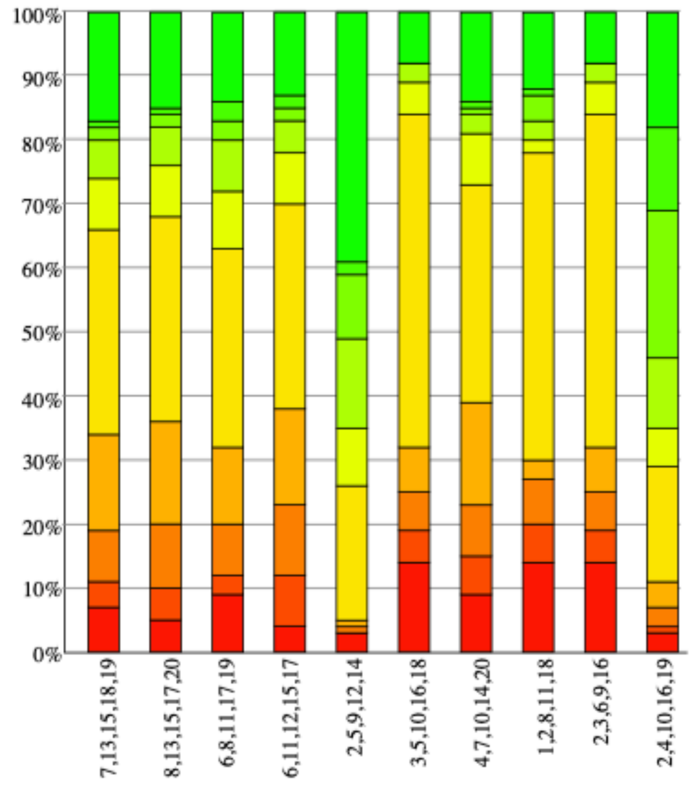


# Evaluation

- Study 2: Multiple-fault Versions

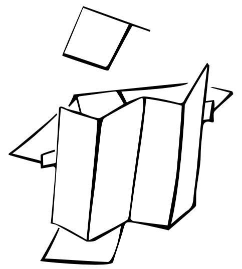


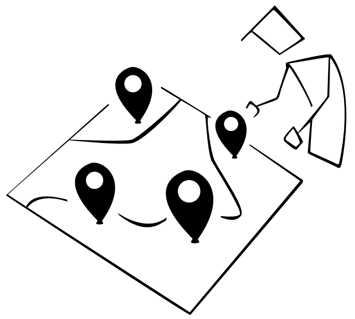
5–fault versions



## Outline

- Problem statement
- Background knowledge
- Method
- Evaluation
- **Related work**
- Future work
- Conclusion
- Discussion questions

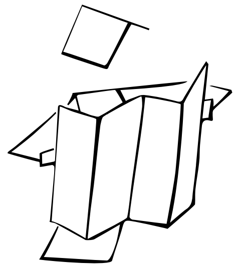


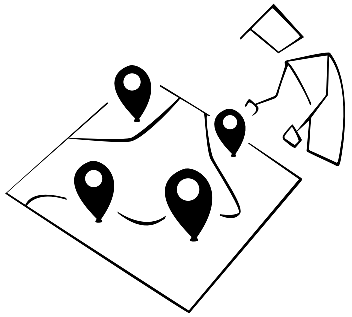


# Related Work

- SeeSoft system
  - Display properties of large amounts of code
  - Zoomed away  perspective
  - Display coverage information
- χSlice
  - Colors statements in a program
  - Show statements participation in passed/failed test cases

# Outline

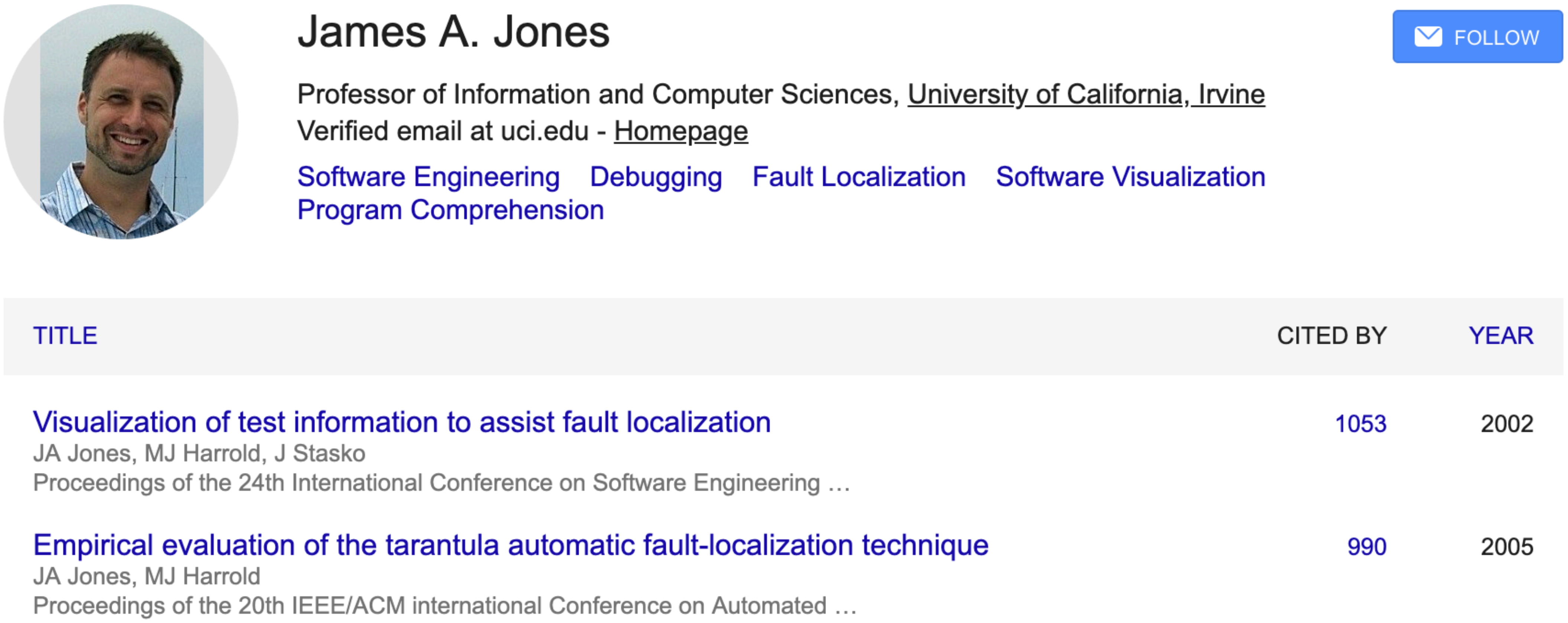




## Future Work

- Technique improvements
  - Include brightness
- Interesting cases investegation
  - Eg. faulty statement may pass when also executing another faulty statement that happens to mask the effects of the first

# Future Work



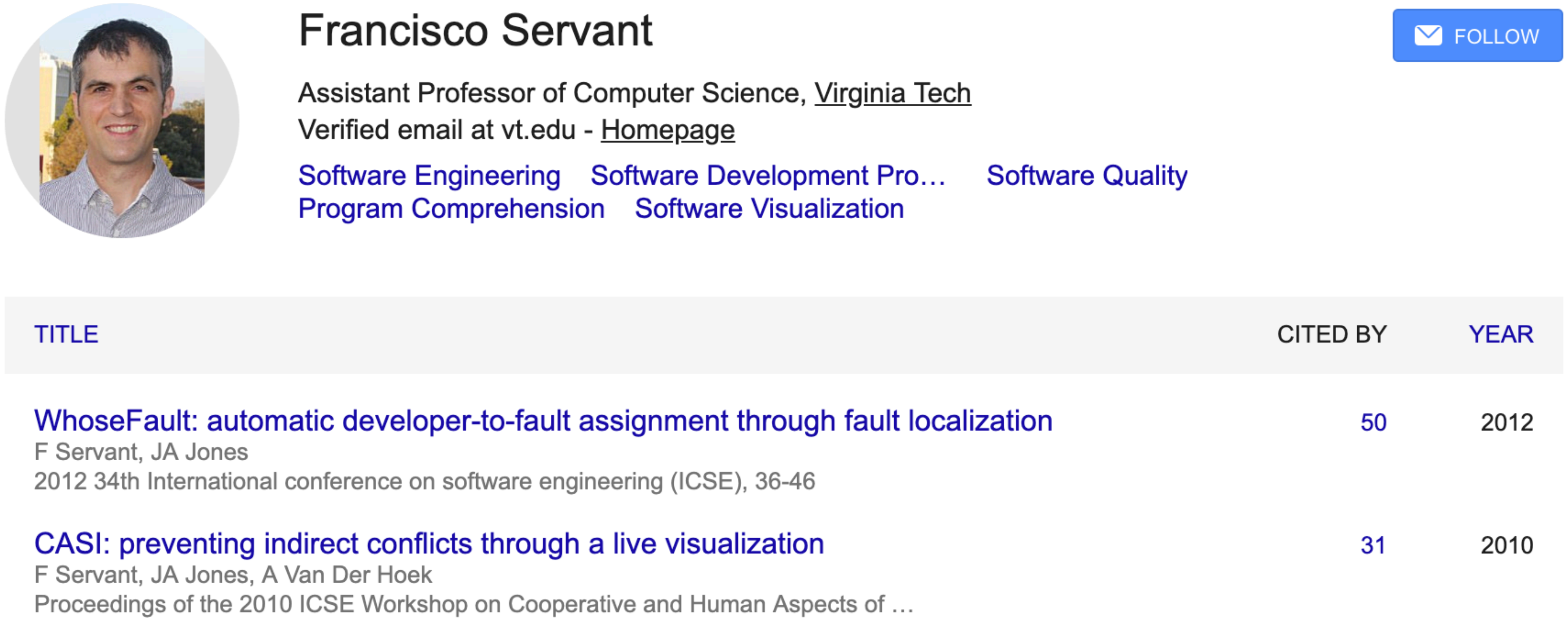
James A. Jones

Professor of Information and Computer Sciences, University of California, Irvine
Verified email at uci.edu - Homepage

Software Engineering Debugging Fault Localization Software Visualization Program Comprehension

| TITLE | CITED BY | YEAR |
|---|---|---|
| Visualization of test information to assist fault localization<br>JA Jones, MJ Harrold, J Stasko<br>Proceedings of the 24th International Conference on Software Engineering … | 1053 | 2002 |
| Empirical evaluation of the tarantula automatic fault-localization technique<br>JA Jones, MJ Harrold<br>Proceedings of the 20th IEEE/ACM international Conference on Automated … | 990 | 2005 |

# Future Work



Francisco Servant

FOLLOW

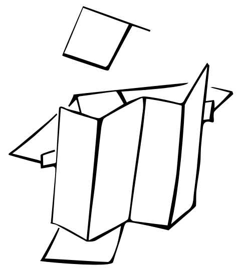
Assistant Professor of Computer Science, Virginia Tech
Verified email at vt.edu - Homepage

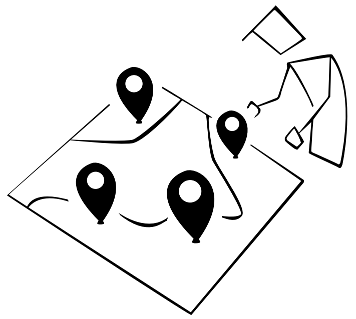
Software Engineering  Software Development Pro…  Software Quality
Program Comprehension  Software Visualization

| TITLE | CITED BY | YEAR |
|---|---|---|
| WhoseFault: automatic developer-to-fault assignment through fault localization<br>F Servant, JA Jones<br>2012 34th International conference on software engineering (ICSE), 36-46 | 50 | 2012 |
| CASI: preventing indirect conflicts through a live visualization<br>F Servant, JA Jones, A Van Der Hoek<br>Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of … | 31 | 2010 |

# Outline

- Problem statement
- Background knowledge
- Method
- Evaluation
- Related work
- Future work
- **Conclusion**
- Discussion questions



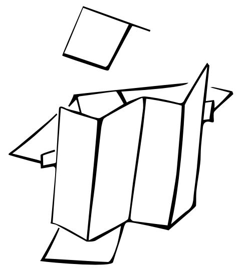


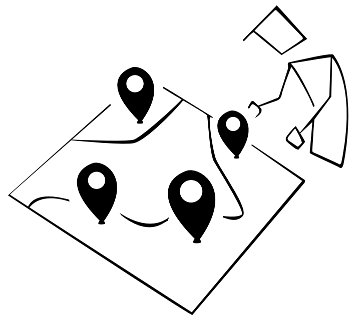
# Conclusion

- Problem
  - Fault localization
    - Reducing the number of delivered faults
- Method
  - Visualization (Tarantula)
    - global view of the test suite execution
- Evaluation
  - Effectiveness
    - False negatives and positives
  - Two studies
    - Single-fault Versions
    - Multiple-fault Versions

# Outline

- Problem statement
- Background knowledge
- Method
- Evaluation
- Related work
- Future work
- Conclusion
- **Discussion questions**





# Discussion Questions

- Approach
  - What are the weaknesses/limitations of the presented method?
  - How can we improve the presented method?
  - How practical is the presented method?
- Evaluation
  - What do you think of the evaluation and evaluation results?
  - What other studies could be preformed to evaluate the work?
- Implications
  - What is next?
    - Do you know of any work based on this paper?

## Thank You