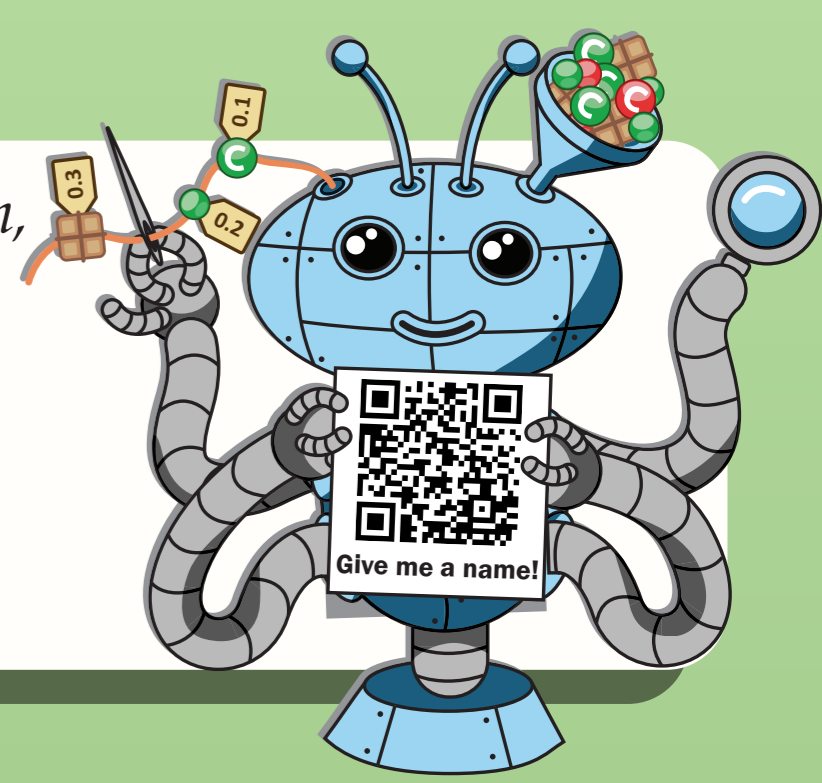




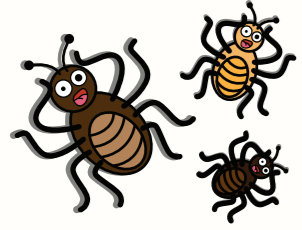
# Aiding Java Developers with Interactive Fault Localization in Eclipse IDE

Gergő Balogh, Ferenc Horváth, Árpád Beszédes

Department of Software Engineering, University of Szeged, Hungary



## Developer's Actions



The user investigates the recommended elements.

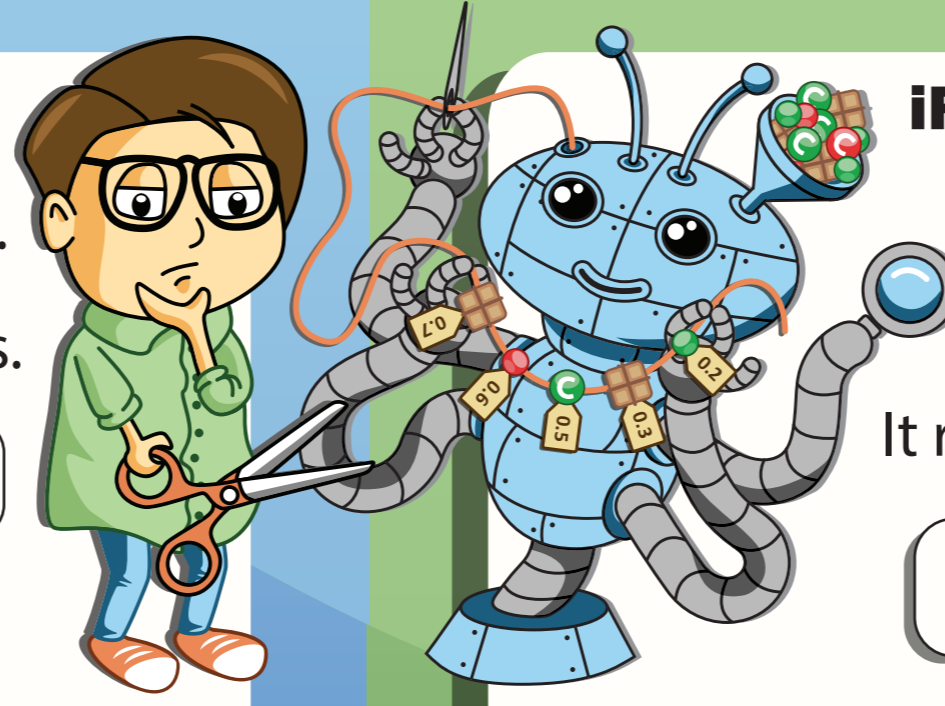
He/She gives one of the provided answers.

Item is not faulty, fault is the context.

Item is not faulty, neither its context!?

Fault is found!

I don't know...



He/She may change and re-run the software to better understand the causes of the error.

## iFL Algorithm's Responses

It calculates an initial SBFL rank.

The elements are shown to the user.

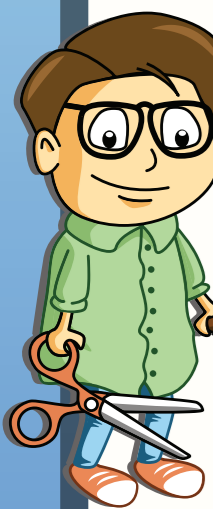
It recalculates the ranking and shows the updated list.

Fault is found: process terminates.

User is not sure: take the next item.

In other cases: I adjust the scores.

## Challenges



Do not generate unnecessary overhead by disturbing the typical workflow of the developers.

Tools should be extensible to make it possible to integrate various already existing SBFL algorithms and future iFL algorithm variants.

Researchers have to define a set of meaningful options to select from.

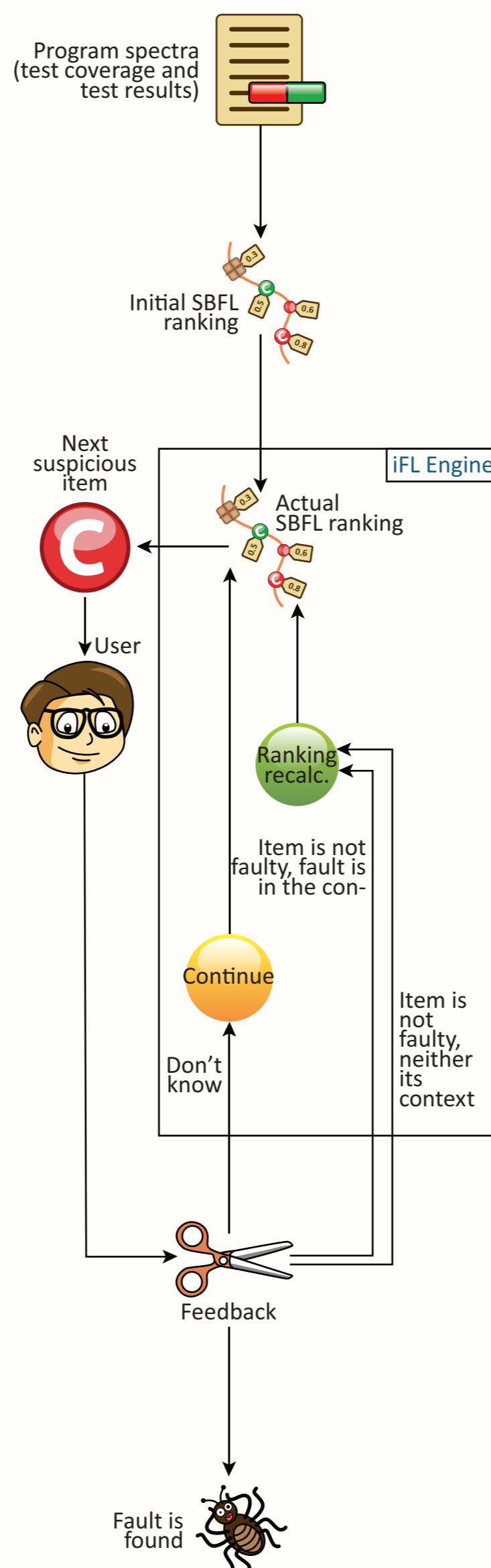
Appropriate actions for various kinds of feedback and relevant code entity context should be defined.

## Problem

Spectrum-Based Fault Localization methods are popular due to their relative simplicity to implement.

Studies highlighted some barriers to the adoption of SBFL in practical settings.

**Goal**  
Increase the practical usefulness of SBFL tools.



## Solution

The developer has additional information about the system of which the SBFL engine is not aware.

Involve interactivity between the user and the FL algorithm.



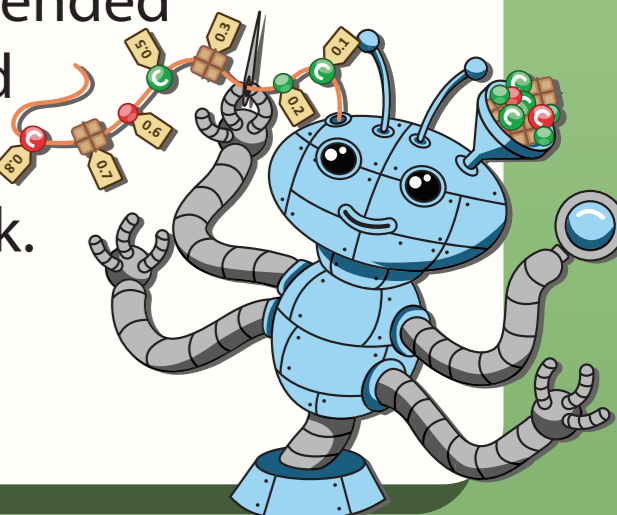
## Achievements

The knowledge of the user is exploited in the ranked list, with which larger code entities can be repositioned in their suspiciousness.

The process starts by calculating an initial rank based on some traditional SBFL.

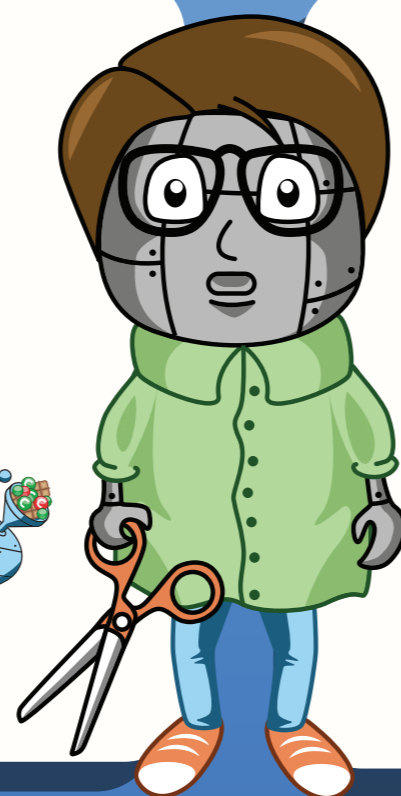
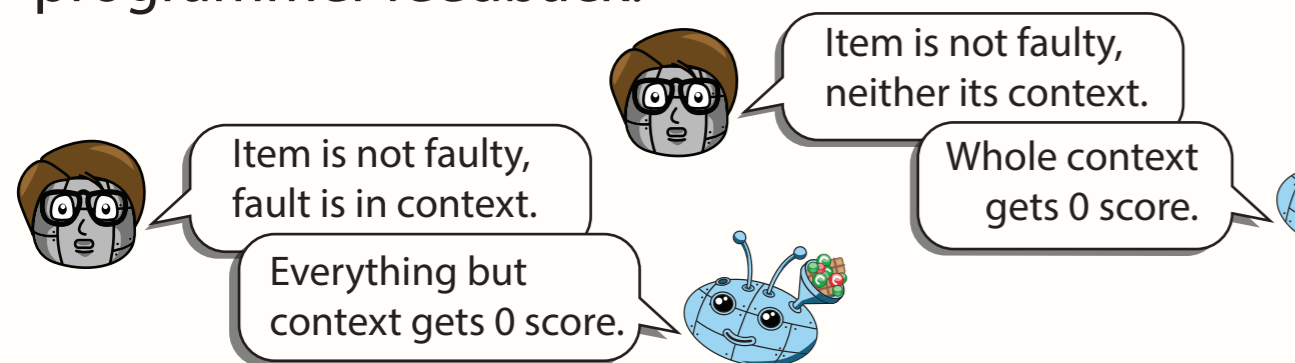
The elements are then shown to the user, and the SBFL engine is waiting for user feedback.

The user investigates the recommended item and gives a feedback.

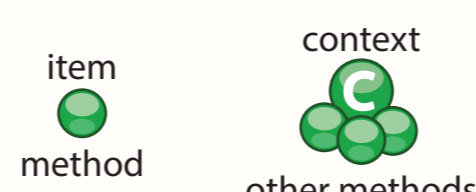


## Simulation of User Actions

We implemented the approach to handle Java systems using simulated users instead of real programmer feedback.



Goal to have a preliminary view of expected improvement in fault localization effectiveness of iFL.



## Experimental Results

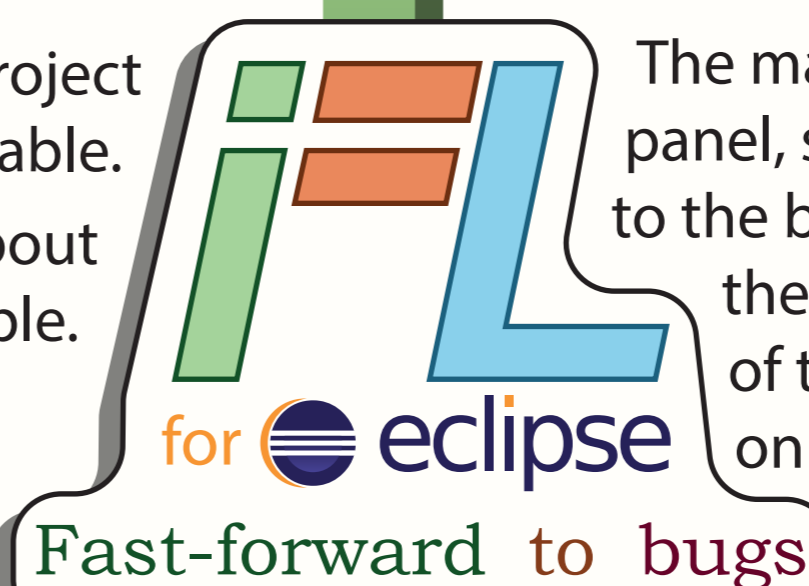
Program	Tarantula	iFL	Diff.	Impr.
commons-lang	3.81 (0.19%)	2.00 (0.09%)	-1.81 (-0.10%)	47.50%
commons-math	7.88 (0.17%)	7.21 (0.15%)	-0.67 (-0.02%)	8.50%
joda-time	17.56 (0.43%)	4.70 (0.12%)	-12.86 (-0.31%)	73.23%
Average	9.75 (0.26%)	4.64 (0.12%)	-5.11 (-0.14%)	43.08%

## iFL for Eclipse

It is an Eclipse plug-in for supporting iFL for Java projects. It reads the tree of project elements and lists them in a table.

0.0000	b	#Sample.ChainFLExample.b()VV	ChainFLExample
0.0000	x	#Sample.ChainFLExample.x()I	ChainFLExample
und-fined	M	#Sample.ChainFLExampleTest.M()V	ChainFLExampleTest
0.00		Is faulty	ChainFLExample
		Neither the subject, nor its neighbours are faulty	ChainFLExampleTest
		Not faulty	ChainFLExampleTest
		Navigate to selected	ChainFLExampleTest
		Navigate to context	ChainFLExampleTest

User sends feedback to the FL engine about the next element in the table.



## Plugin Architecture

The main UI is an Eclipse part, a graphical panel, serving as the front end. It is connected to the back end components, whose purpose is the update of scores and the recalculation of the rank list based on user input.

