Mining Performance Regression Inducing Code Changes in Evolving Software

Qi Luo, Denys Poshyvanyk, Mark Grechanik*

College of William and Mary
*University of Illinois at Chicago

MSR 2016
Austin, TX, U.S.


public synchronized void regressionFunc ();
static Regressions staticRT = new Regressions();
public void setTeams(Collection<Team> teams) {
this.teams = teams;
Backlog.staticRT.regressionFunc();
Set<Integer> selectedBacklogIds = this.getSelectedBacklogs();
if(selectedBacklogIds == null || selectedBacklogIds.size() == 0) {
Collection<Product> products = new ArrayList<Product>();
productBusiness.storeAllTimeSheets(products);
for (Product product: products) {
selectedBacklogIds.add(product.getId());
}
}
return Action.SUCCESS;
for(Story child : story.getChildren()) {
if (child.getId() == story.getId()) {
continue;
}
StoryTreeBranchMetrics childMetrics = this.calculateStoryTreeMetrics(child
}
return metrics;
60
5
10
15
20
25
30
35
40
45
50
55

Automated Detection of Performance Regressions Using Regression Models on Clustered Performance Counters

Weiyi Shang, Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
Queen's University, Kingston, Ontario, Canada
{swy, ahmed}@cs.queensu.ca

Mohamed Nasser, Parminder Flora
BlackBerry
Waterloo, Ontario, Canada

ABSTRACT

Performance testing is conducted before deploying system updates in order to ensure that the performance of large software systems did not degrade (i.e., no performance regressions). During such testing, thousands of performance counters are collected. However, comparing thousands of performance counters across versions of a software system is very time consuming and error-prone. In an effort to auto-

1. INTRODUCTION

Performance assurance activ... the release cycle of large softwa... activities aim to identify and el... sions in each newly released v... mance regressions are response t... expected resource utilization a... gressions may compromise the

Main Effects Screening: A Distributed Continuous Quality Assurance Process for Monitoring Performance Degradation in Evolving Software Systems

Cemal Yilmaz†, Arvind S. Krishna‡, Atif Memon†, Adam Porter†, Douglas C. Schmidt‡, Aniruddha Gokhale‡, Balachandran Natarajan‡

†Dept. of Computer Science, University of Maryland, College Park, MD 20742

‡Dept. of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN 37235

Abstract

Developers of highly configurable performance- ...stems often use a type of in-house ...d "regression testing" to ensure ...ns have not adversely affected their ...nce across its large configuration ... time and resource constraints often

Although these software parameters promote flexibility and portability, they also require that the software be tested in an enormous number of configurations. This creates serious challenges for developers who must ensure that their decisions, additions, and modifications work across this large (and often changing) configuration space:

- Settings that maximize performance for a particular

Mining Performance Regression Testing Repositories for Automated Performance Analysis

King Chun Foo[1], Zhen Ming Jiang[2], Bram Adams[2], Ahmed E. Hassan[2], Ying Zou[1], Parminder Flora[3]

Department of Electrical and Computer Engineering[1]
Queen's University
Kingston, ON, Canada
{k.foo, ying.zou}@queensu.ca

School of Computing[2]
Queen's University
Kingston, ON, Canada
{zmjiang, bram, ahmed}@cs.queensu.ca

Performance Engineering[3]
Research In Motion
Waterloo, ON, Canada

Abstract— Performance regression testing detects performance regressions in a system under load. Such regressions refer to situations where software performance degrades compared to previous releases, although the new version behaves correctly. In current practice, performance analysts must manually analyze performance regression testing data to uncover performance regressions. This process is both time-consuming and error-prone due to the large volume of metrics collected, the absence of formal performance objectives and the subjectivity of individual

During the course of the te... the running system is re... utilizations). After each tes... domain-knowledge and pri... deviations of metric values... test. A defect report will b... concludes that the observed... regressions. In a large en...

Automated Detection of Performance Regressions Using Statistical Process Control Techniques

Thanh H. D. Nguyen, Bram Adams, Zhen Ming Jiang, Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
School of Computing, Queen's University
Kingston, Ontario, Canada
{thanhnguyen,bram,zmjiang,ahmed}@cs.queensu.ca

Mohamed Nasser, Parminder Flora
Performance Engineering
Research In Motion (RIM)
Waterloo, Ontario, Canada

ABSTRACT

The goal of performance regression testing is to check for performance regressions in a new version of a software system. Performance regression testing is an important phase in the software development process. Performance regression testing is very time consuming yet there is usually little time assigned for it. A typical test run would output

features, code changes might degrade the software's performance. Hence, performance engineers must perform regression tests to make sure that the software still performs as good as previous versions. Performance regression testing is very important to large software systems where a large number of field problems are performance related [19].

Performance regression testing is very time consuming yet

V_i

```
public synchronized
        void calculate();

input a, b
A item;

item = new A();

item.calculate();
```

V_{i+1}

```
public synchronized
        void calculate();

input a, b
A item;
if (a > b)
        item = new A();
else item = A.getItem();
item.calculate();
```

Increased Execution Time

V_i

```
public synchronized
        void calculate();

input a, b
A item;

item = new A();

item.calculate();
```

V_{i+1}

```
public synchronized
        void calculate();

input a, b
A item;
if (a > b)
        item = new A();
else item = A.getItem();
item.calculate();
```

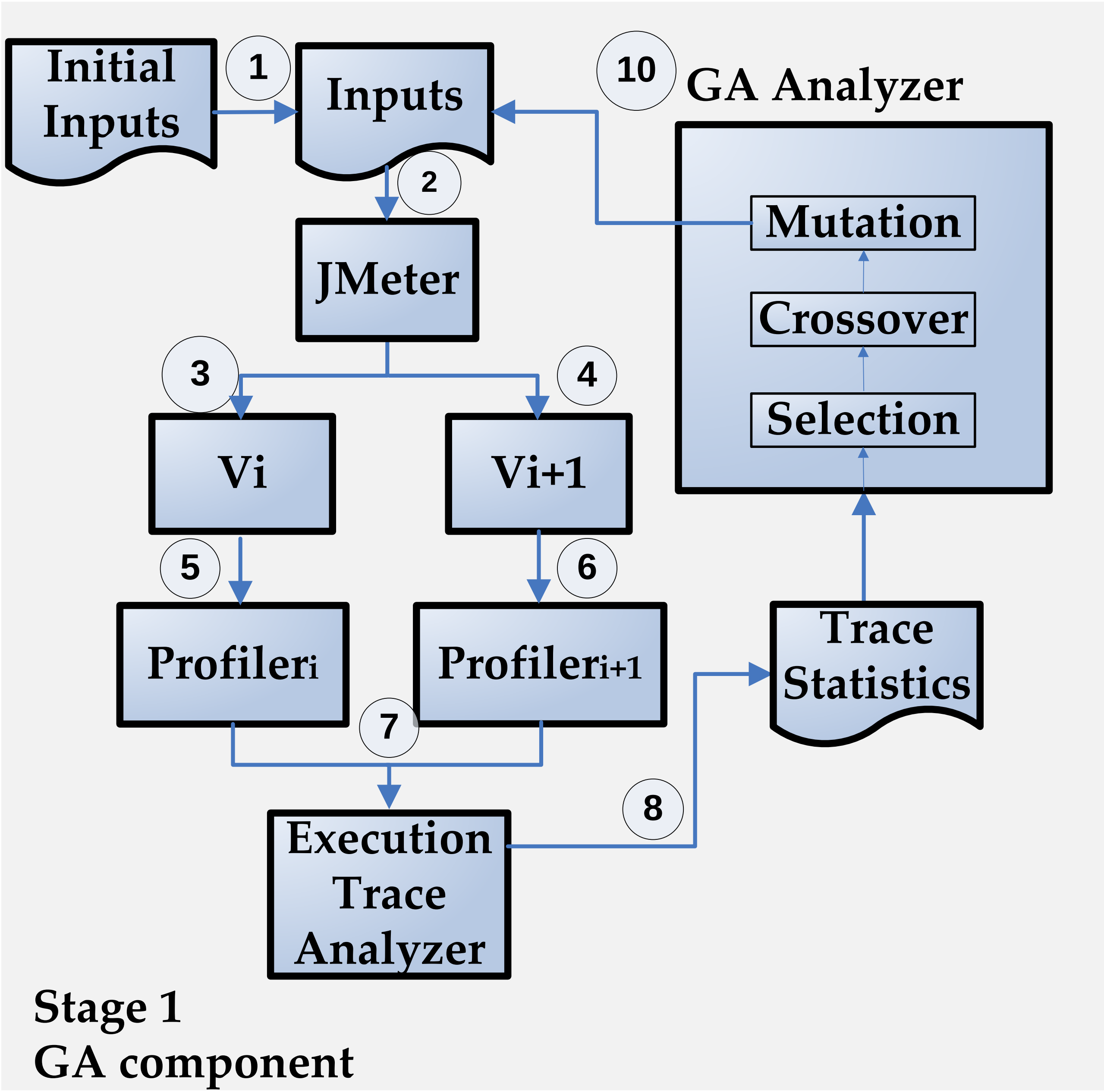
Increased Execution Time

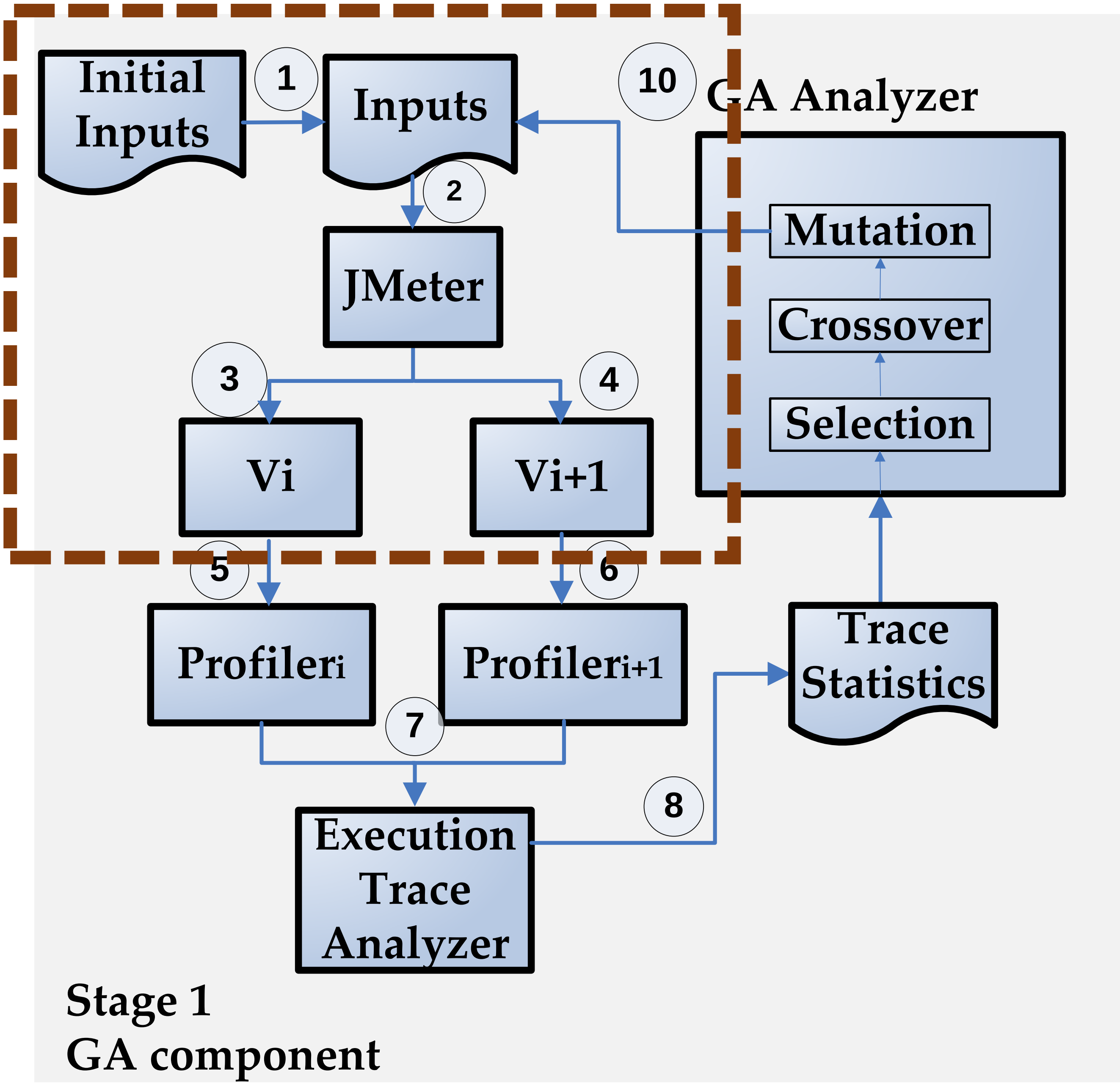
Problematic Code Change


public synchronized void regressionFunc ();
static Regressions staticRT = new Regressions();
public void setTeams(Collection<Team> teams) {
this.teams = teams;
Backlog.staticRT.regressionFunc();
Set<Integer> selectedBacklogIds = this.getSelectedBacklogs();
if(selectedBacklogIds == null || selectedBacklogIds.size() == 0) {
Collection<Product> products = new ArrayList<Product>();
productBusiness.storeAllTimeSheets(products);
for (Product product: products) {
selectedBacklogIds.add(product.getId());
}
}
return Action.SUCCESS;
for(Story child : story.getChildren()) {
if (child.getId() == story.getId()) {
continue;
}
StoryTreeBranchMetrics childMetrics = this.calculateStoryTreeMetrics(child
}
return metrics;
60
5
10
15
20
25
30
35
40
45
50
55

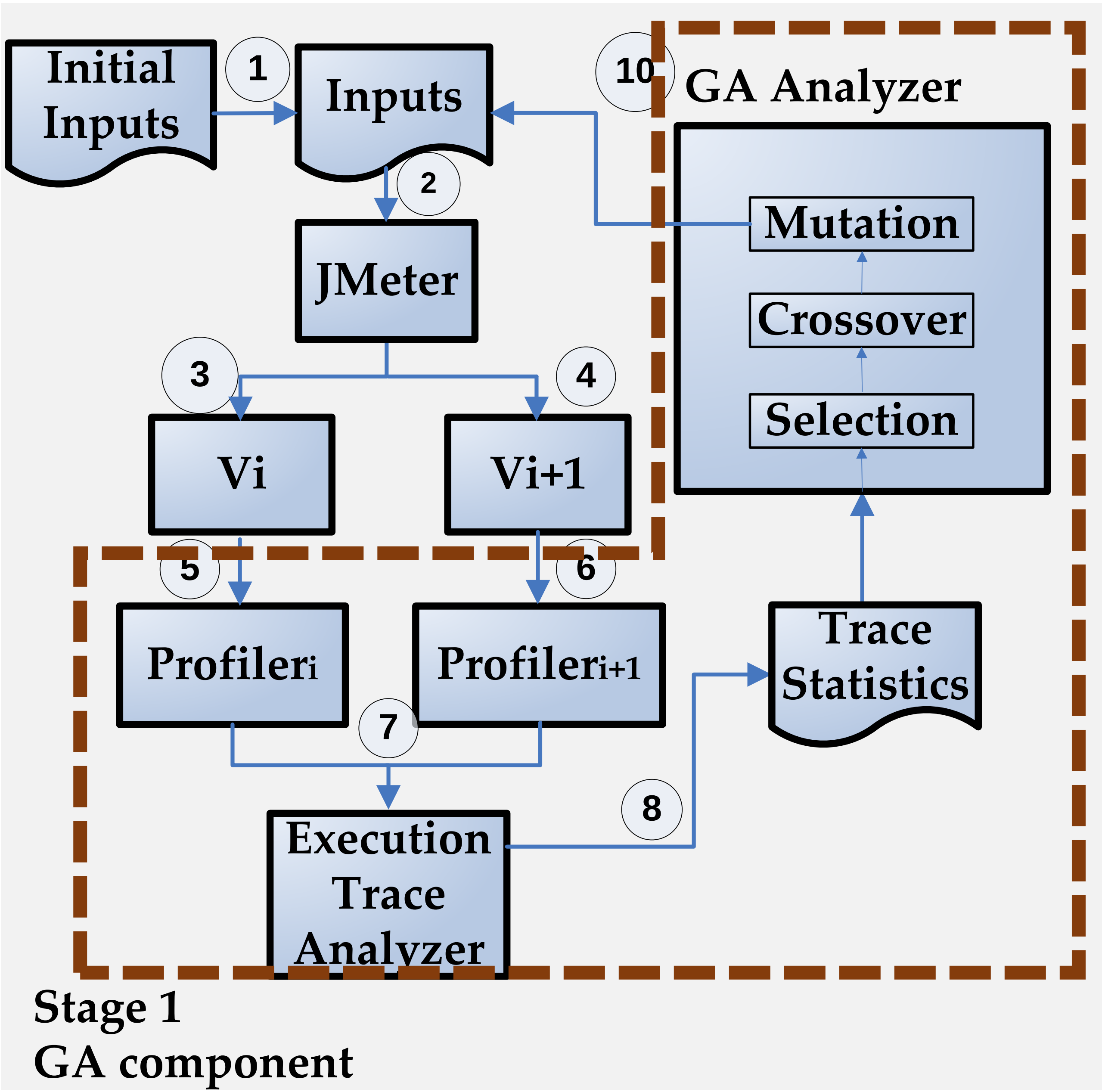
PerfImpact – performance regression testing

- Search-based input profiling to find specific inputs for exposing performance regressions
- Change impact analysis (CIA) to recommend the problematic code changes

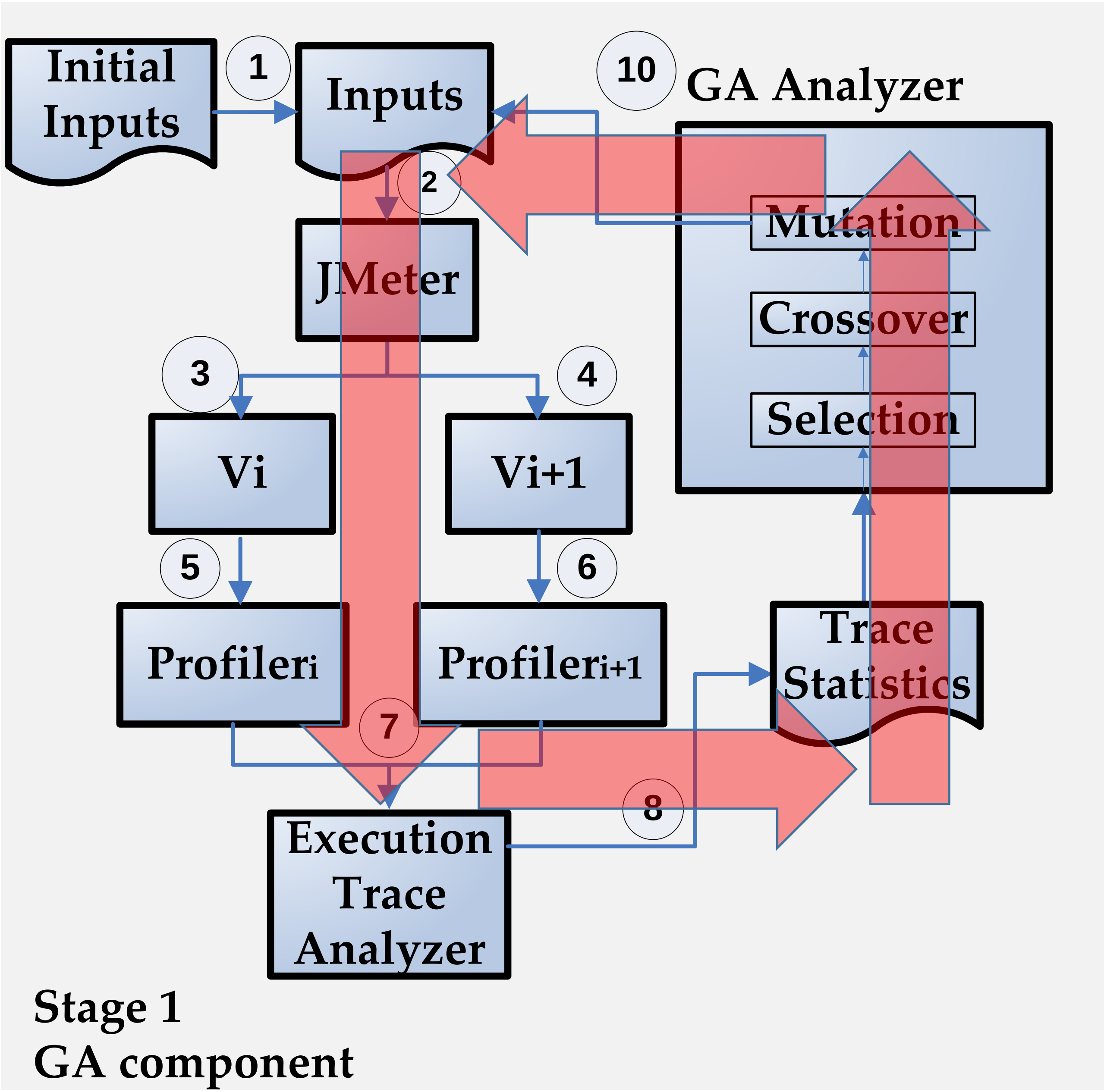

Initial Inputs
1
Inputs
10
GA Analyzer
2
JMeter
Mutation
Crossover
Selection
3
4
Vi
Vi+1
5
6
Profileri
Profileri+1
Trace Statistics
7
8
Execution Trace Analyzer
Stage 1
GA component



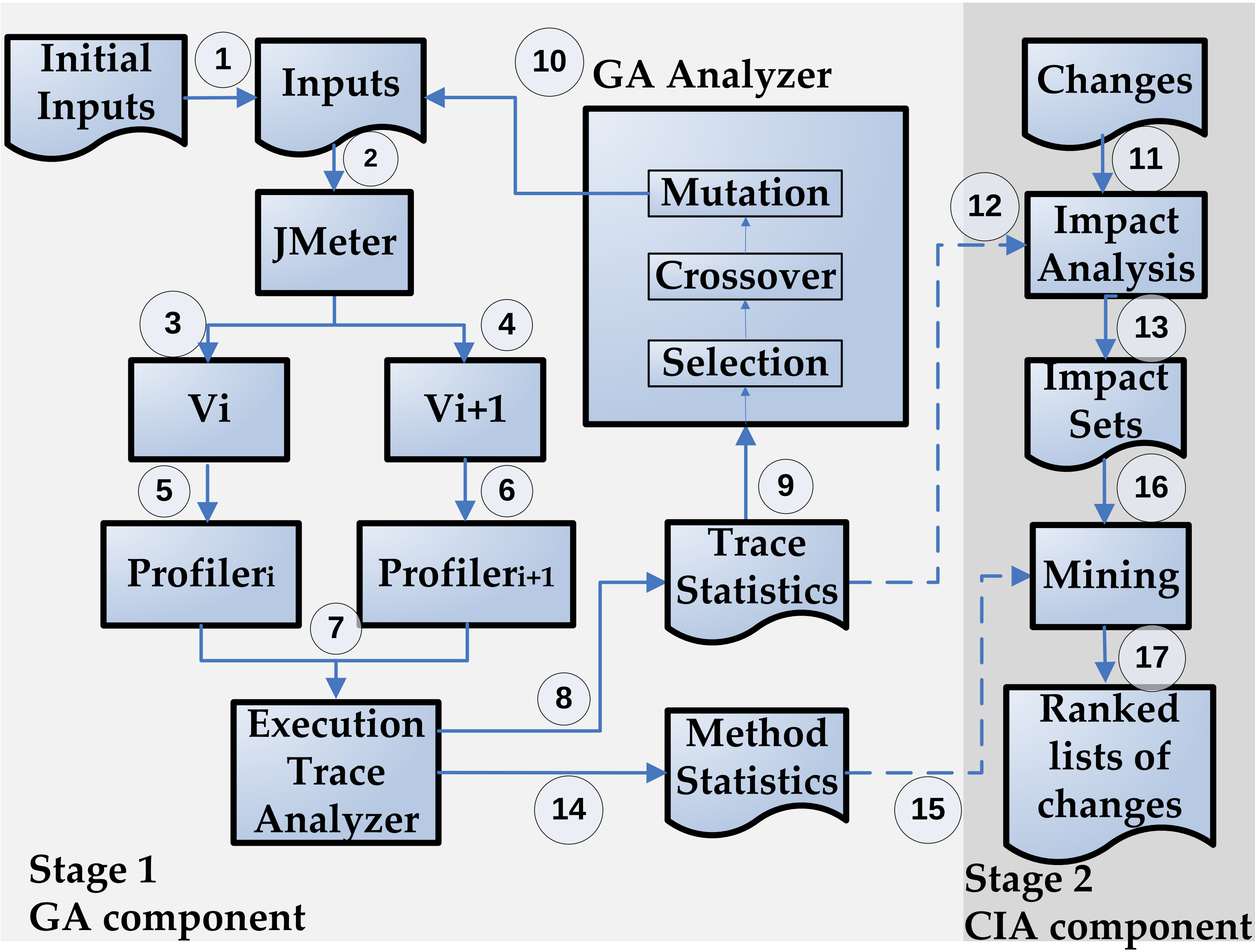
Initial Inputs
1
Inputs
10
GA Analyzer
2
JMeter
Mutation
Crossover
Selection
3
4
Vi
Vi+1
5
6
Profileri
Profileri+1
Trace Statistics
7
8
Execution Trace Analyzer
Stage 1
GA component

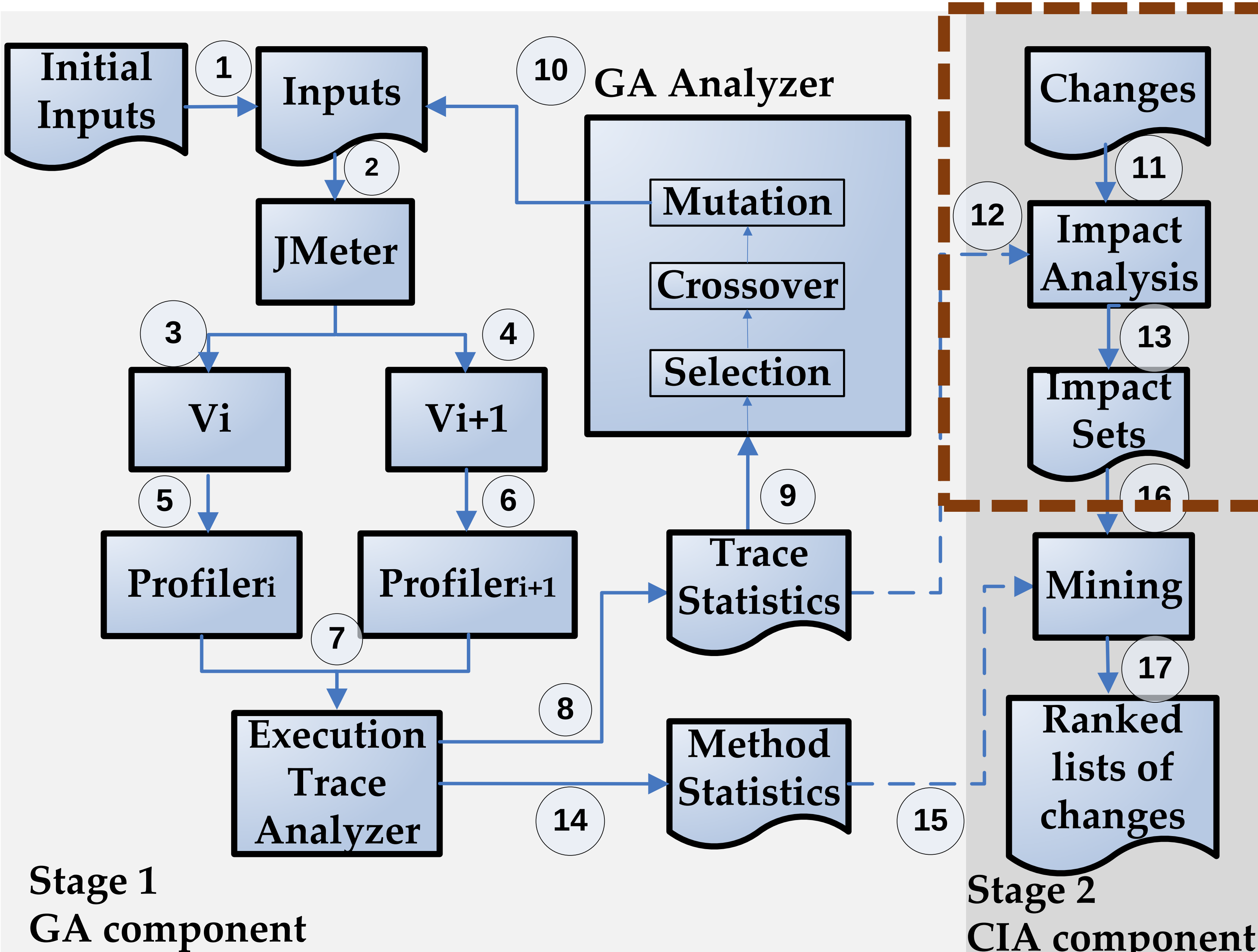


Initial Inputs
1
Inputs
2
JMeter
3
4
Vi
Vi+1
5
6
Profileri
Profileri+1
7
Execution Trace Analyzer
8
Trace Statistics
10
GA Analyzer
Mutation
Crossover
Selection
Stage 1
GA component

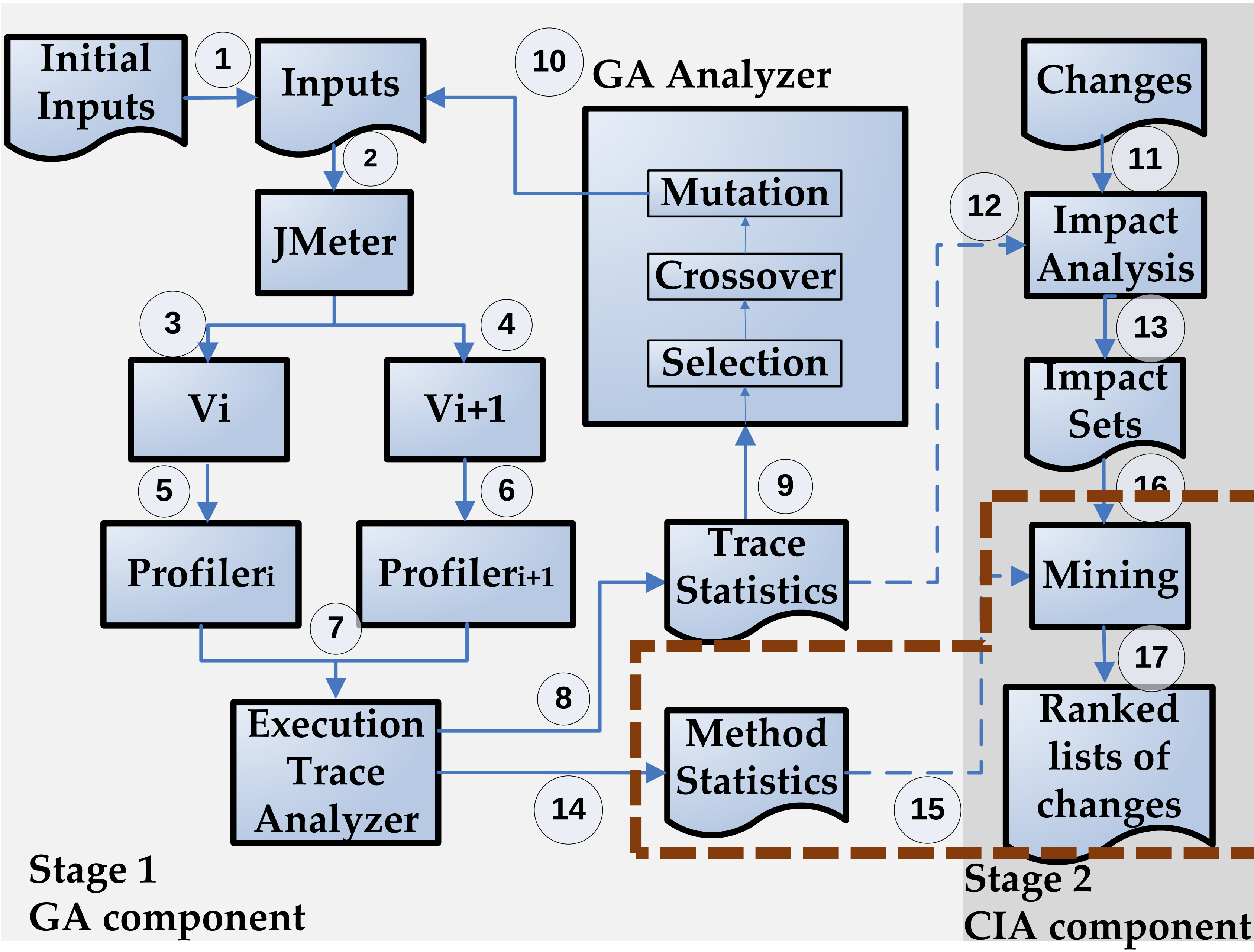


Initial Inputs
1
Inputs
2
JMeter
3
4
Vi
Vi+1
5
6
Profileri
Profileri+1
7
Execution Trace Analyzer
8
Trace Statistics
10
GA Analyzer
Mutation
Crossover
Selection
Stage 1
GA component

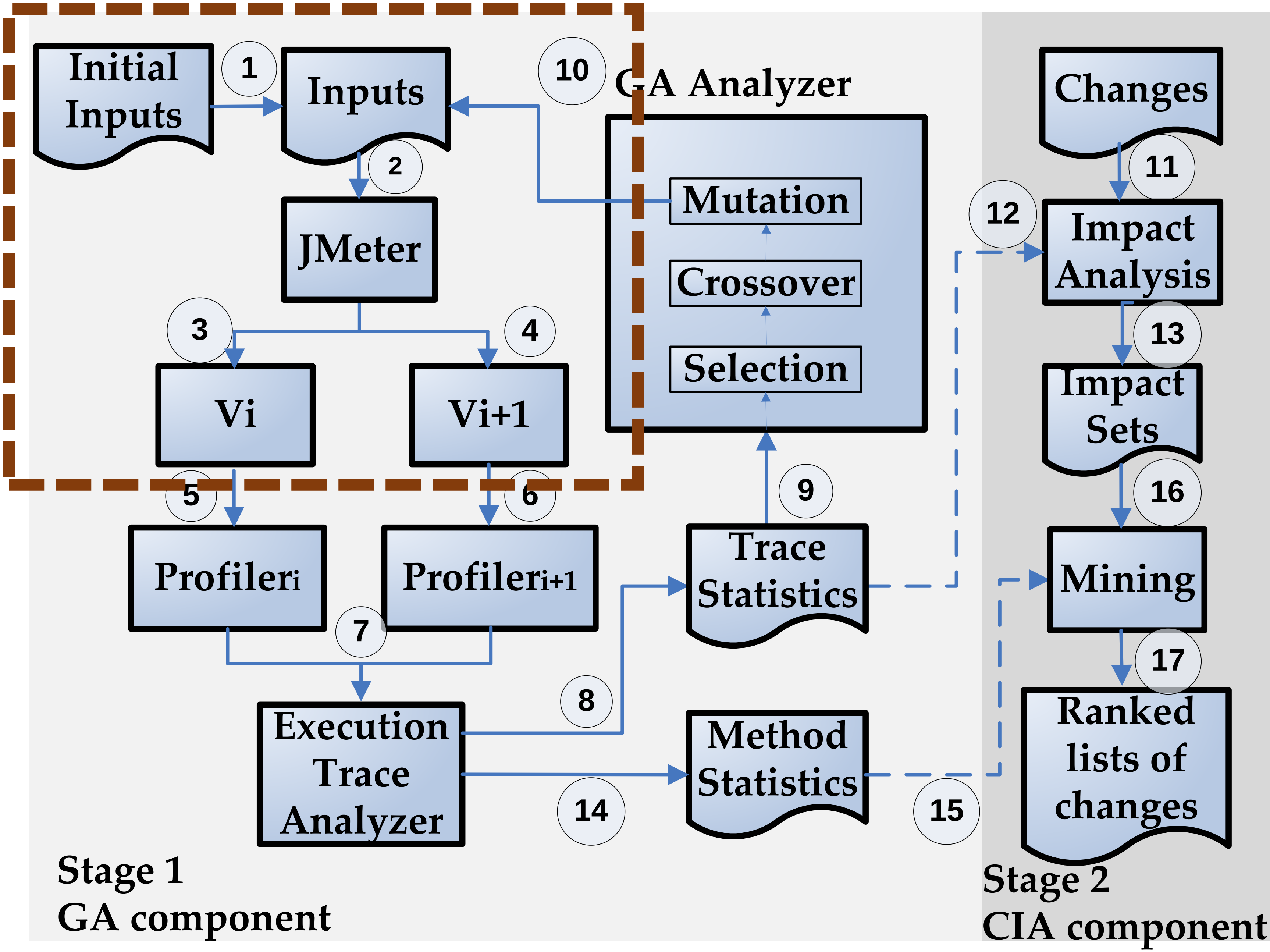

Initial Inputs
1
Inputs
2
JMeter
3
4
Vi
Vi+1
5
6
Profileri
Profileri+1
7
Execution Trace Analyzer
8
14
10
GA Analyzer
Mutation
Crossover
Selection
9
Trace Statistics
Method Statistics
15
Changes
11
12
Impact Analysis
13
Impact Sets
16
Mining
17
Ranked lists of changes
Stage 1
GA component
Stage 2
CIA component

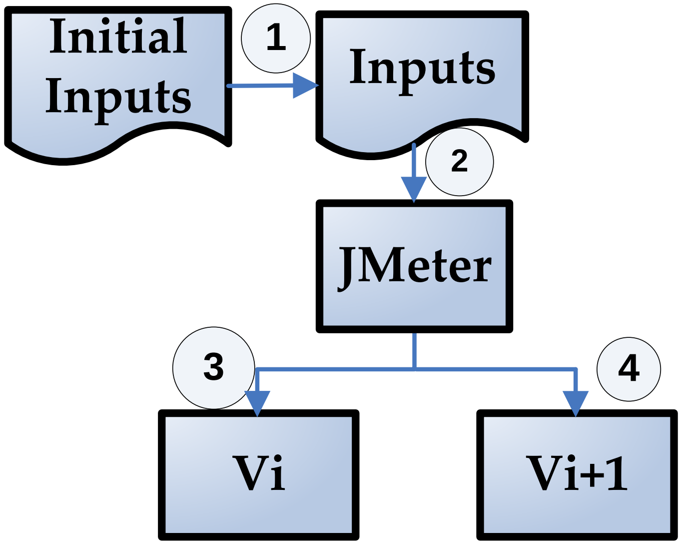


Initial Inputs
1
Inputs
2
JMeter
10
GA Analyzer
Mutation
Crossover
Selection
3
4
Vi
Vi+1
5
6
Profileri
Profileri+1
7
9
Trace Statistics
8
Execution Trace Analyzer
14
Method Statistics
15
Changes
11
12
Impact Analysis
13
Impact Sets
16
Mining
17
Ranked lists of changes
Stage 1
GA component
Stage 2
CIA component



Initial Inputs
1
Inputs
2
JMeter
3
4
Vi
Vi+1
5
6
Profileri
Profileri+1
7
Execution Trace Analyzer
8
14
10
GA Analyzer
Mutation
Crossover
Selection
9
Trace Statistics
Method Statistics
15
Changes
11
12
Impact Analysis
13
Impact Sets
16
Mining
17
Ranked lists of changes
Stage 1
GA component
Stage 2
CIA component


Initial Inputs
1
Inputs
2
JMeter
3
4
Vi
Vi+1
5
6
Profileri
Profileri+1
7
Execution Trace Analyzer
8
14
10
GA Analyzer
Mutation
Crossover
Selection
9
Trace Statistics
Method Statistics
Changes
11
12
Impact Analysis
13
Impact Sets
16
Mining
17
Ranked lists of changes
15
Stage 1
GA component
Stage 2
CIA component

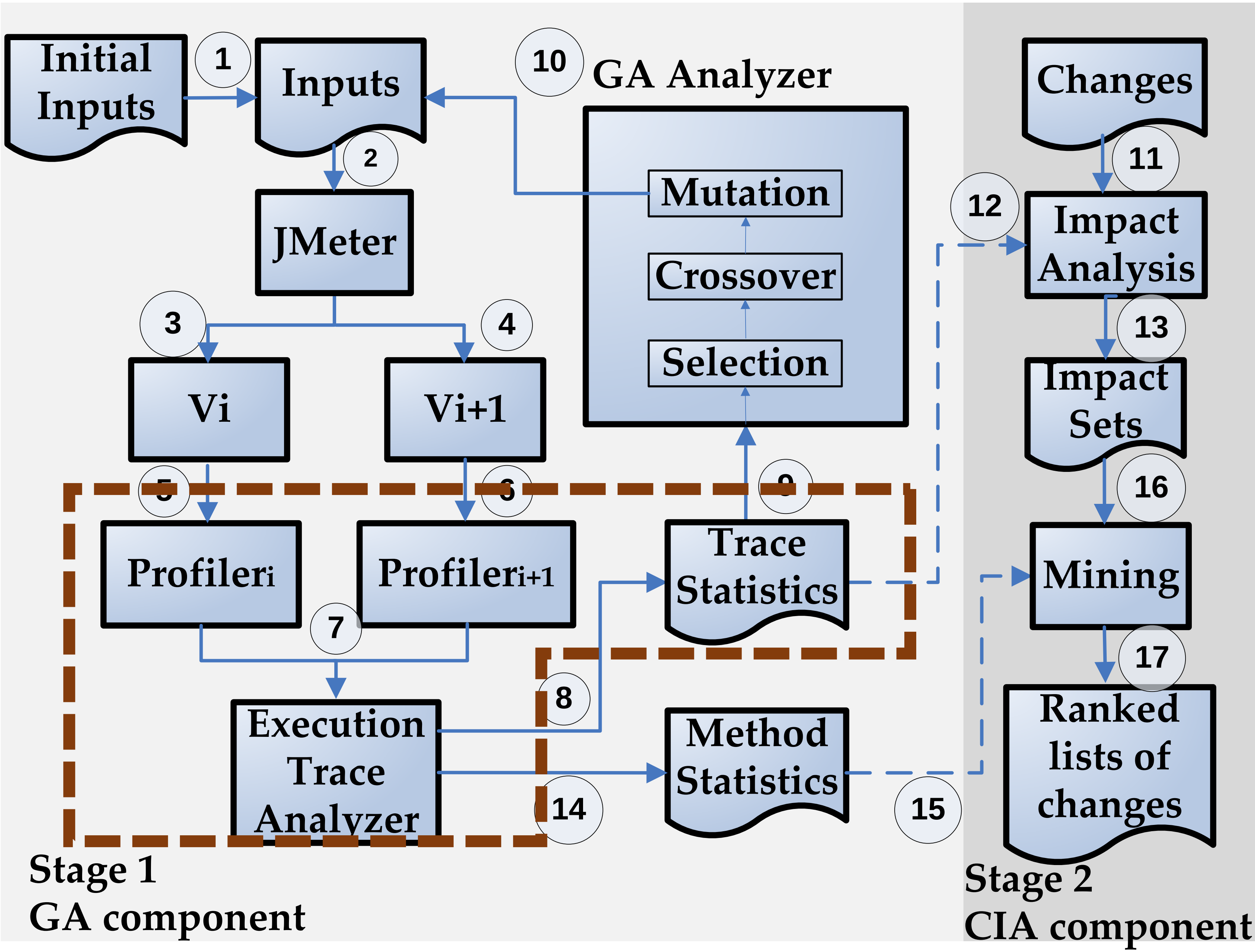


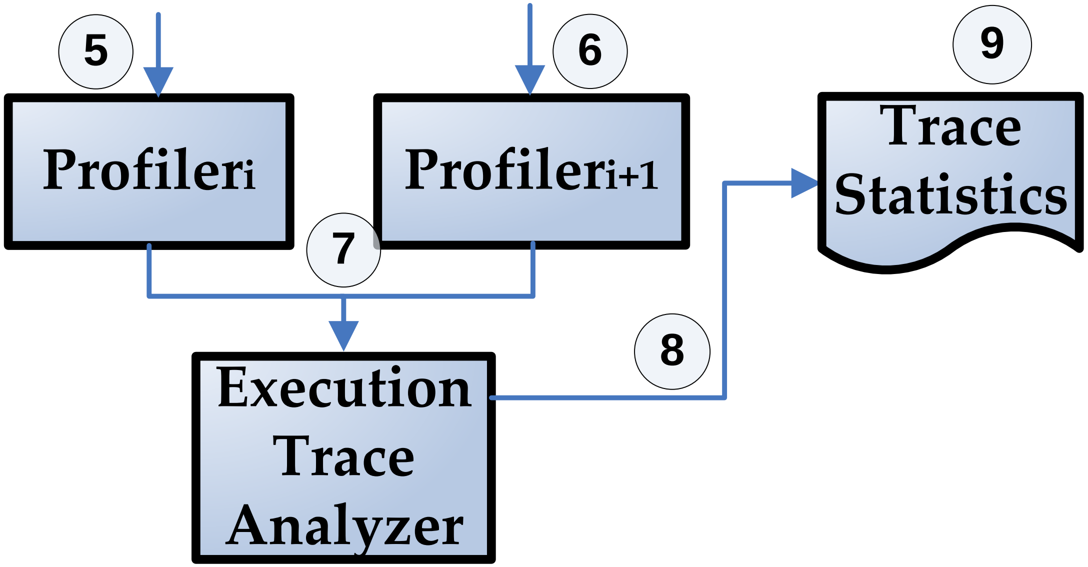
Genes:

Input 1: http://localhost:8080/Agilefant/editUser.action
Input 2: http://localhost:8080/Agilefant/editProduct.action?productId=5
Input 3: http://localhost:8080/Agilefant/editProduct.action?productId=8
……

A chromosome/individual

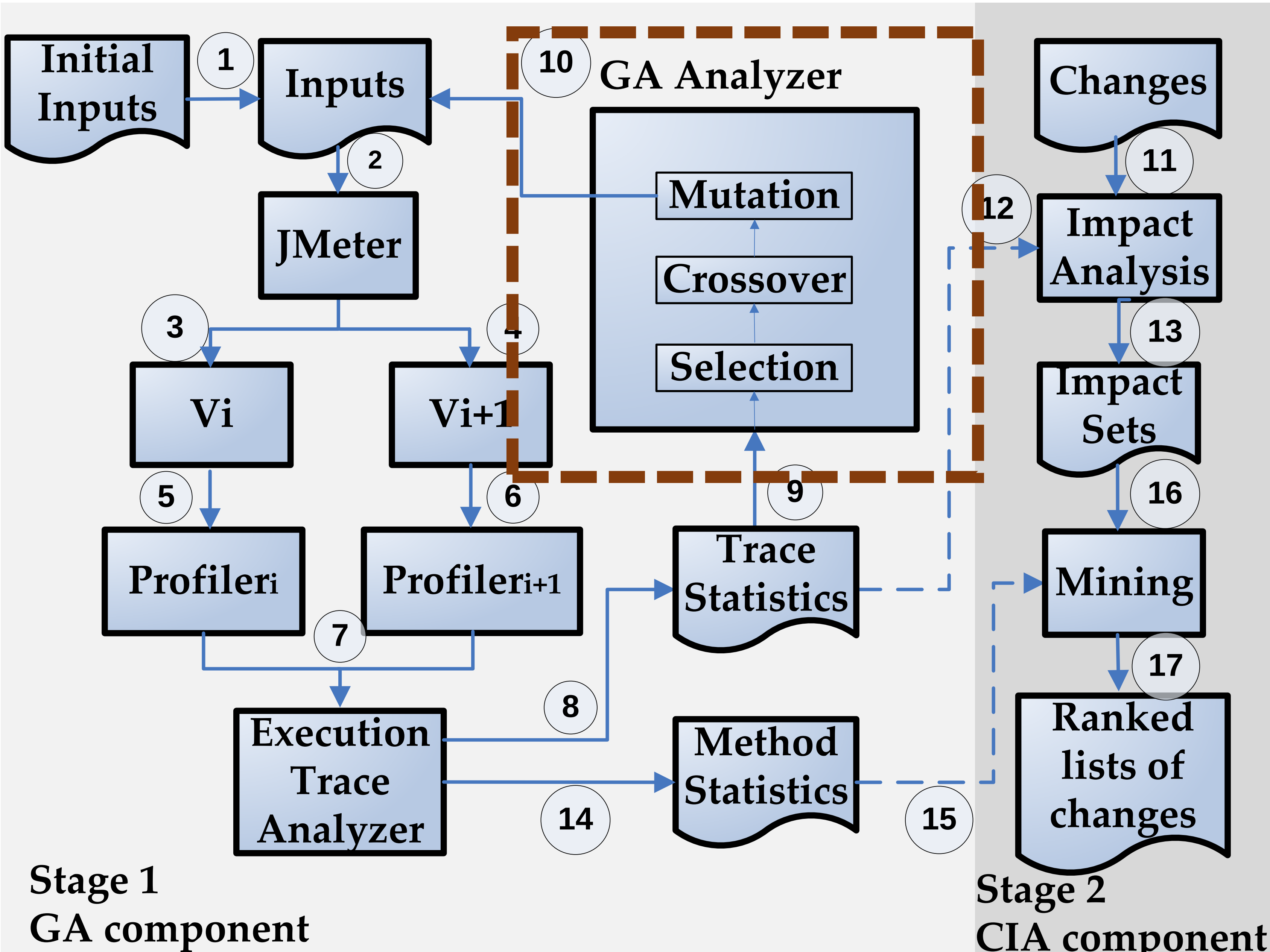
Individual 1: 2, 18, 36, 27, 11, 13, 6, 43, 64, 12, 85, 49, 12, 53, 44, 78, 31, 47


Initial Inputs
1
Inputs
10
GA Analyzer
Changes
2
Mutation
11
JMeter
Crossover
12
Impact Analysis
3
4
Selection
13
Vi
Vi+1
Impact Sets
5
6
9
16
Profileri
Profileri+1
Trace Statistics
Mining
7
17
8
Execution Trace Analyzer
Method Statistics
Ranked lists of changes
14
15
Stage 1
GA component
Stage 2
CIA component

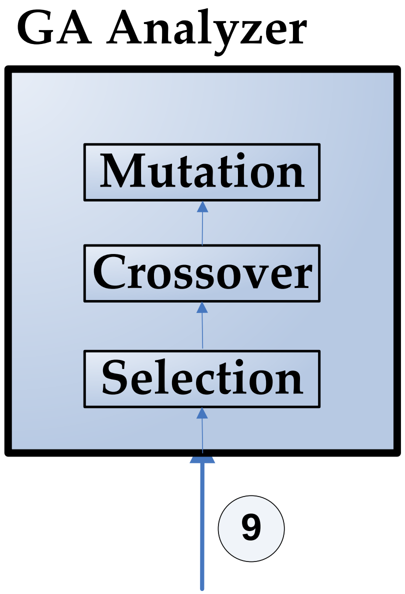


An individual (I_j)

- Execution time in V_i (t_j^i)
- Execution time in V_{i+1} (t_j^{i+1})

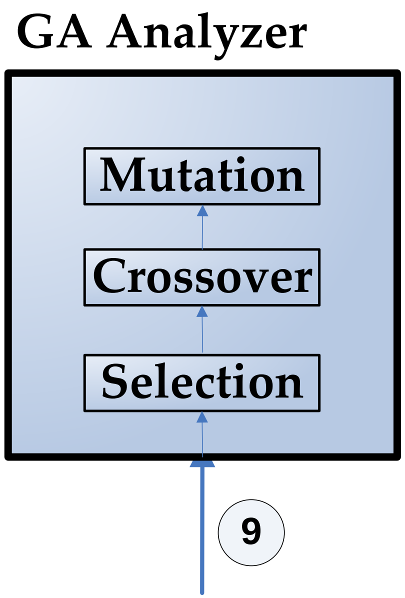


Initial Inputs
1
Inputs
2
JMeter
3
4
Vi
Vi+1
5
6
Profileri
Profileri+1
7
Execution Trace Analyzer
8
14
10
GA Analyzer
Mutation
Crossover
Selection
9
Trace Statistics
Method Statistics
15
Changes
11
12
Impact Analysis
13
Impact Sets
16
Mining
17
Ranked lists of changes
Stage 1
GA component
Stage 2
CIA component



- Selection

  Fitness value for $I_j = t_j^{i+1} - t_j^{i}$



- Selection

  Fitness value for $I_j = t_j^{i+1} - t_j^i$

- Crossover

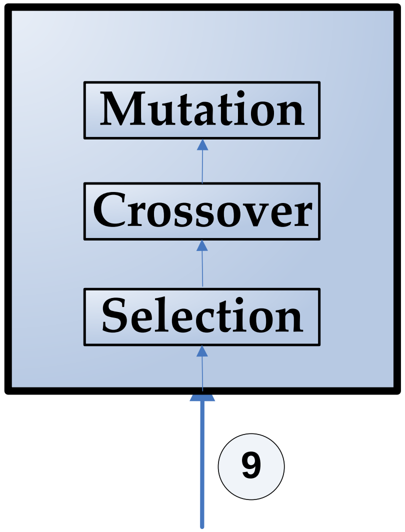
Parent 1: 2, 18, 36, 27, 11, 13, 6, 43, 64, 12, 85, 49, 12, 53, 44, 91, 79, 23, 3, 19

Parent 2: 23, 95, 1, 67, 35, 81, 7, 17, 51, 102, 56, 39, 72, 3, 54, 37, 13, 86, 47, 76

Child 1: 2, 18, 36, 27, 11, 13, 6, 17, 51, 102, 56, 39, 72, 3, 54, 37, 13, 86, 47, 76

Child 2: 23, 95, 1, 67, 35, 81, 7, 43, 64, 12, 85, 49, 12, 53, 44, 91, 79, 23, 3, 19

GA Analyzer



- Selection

  Fitness value for $I_j = t_j^{i+1} - t_j^{i}$

- Crossover

Parent 1: 2, 18, 36, 27, 11, 13, 6, 43, 64, 12, 85, 49, 12, 53, 44, 91, 79, 23, 3, 19

Parent 2: 23, 95, 1, 67, 35, 81, 7, 17, 51, 102, 56, 39, 72, 3, 54, 37, 13, 86, 47, 76

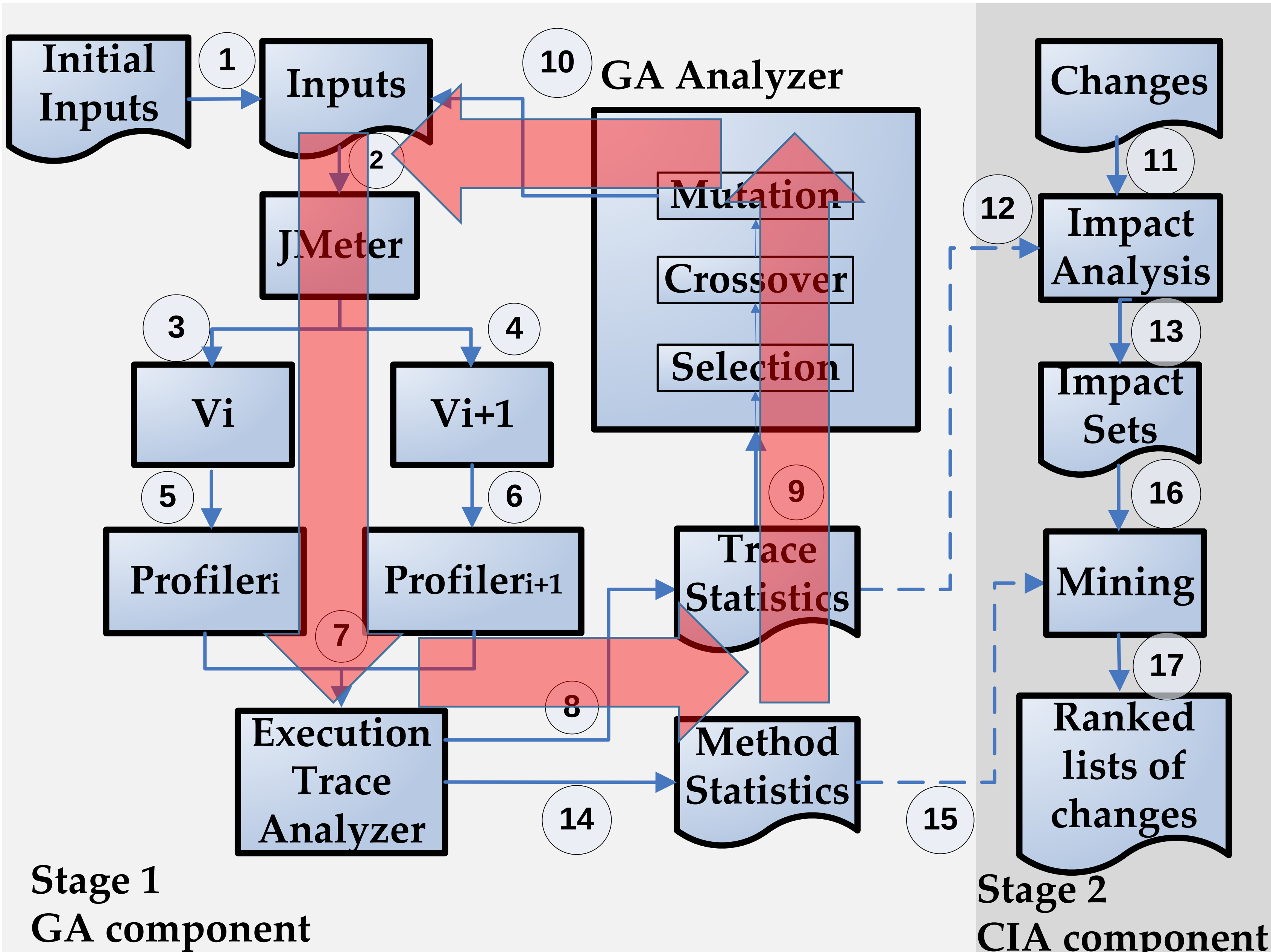
Child 1: 2, 18, 36, 27, 11, 13, 6, 17, 51, 102, 56, 39, 72, 3, 54, 37, 13, 86, 47, 76

Child 2: 23, 95, 1, 67, 35, 81, 7, 43, 64, 12, 85, 49, 12, 53, 44, 91, 79, 23, 3, 19

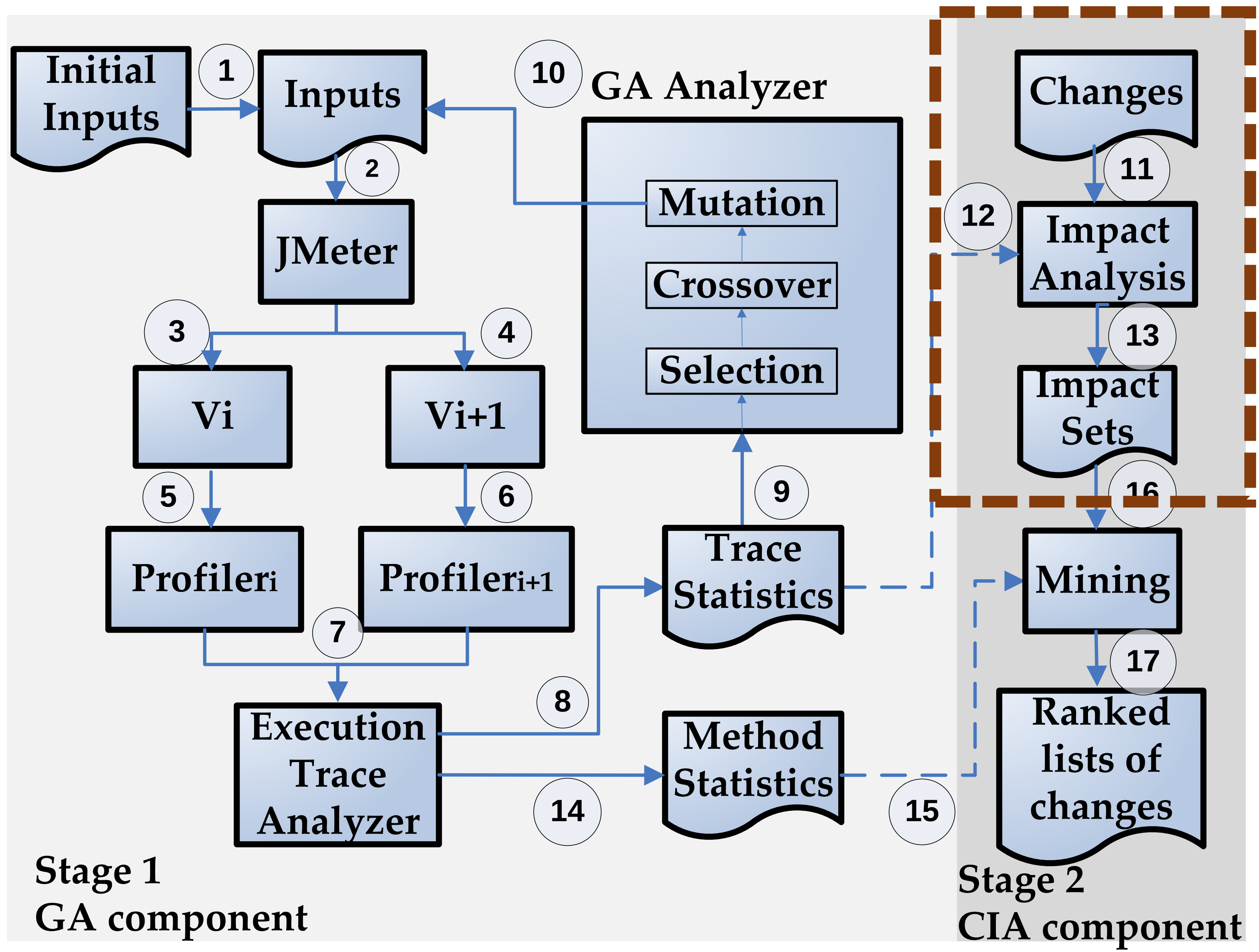
- Mutation

Parent : 2, 18, 36, 27, 11, 13, 6, 43, 64, 12, 85, 49, 12, 53, 44, 91, 79, 23, 3, 19

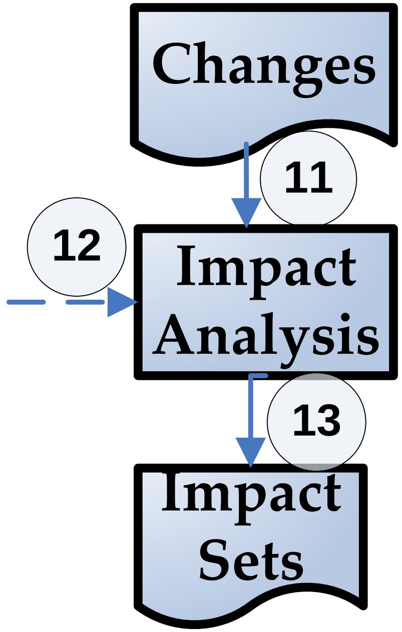
Child : 2, 18, 36, 27, 11, 13, 6, 43, 64, 73, 85, 49, 12, 53, 44, 91, 79, 23, 3, 19



Initial Inputs
1
Inputs
2
JMeter
10
GA Analyzer
Mutation
Crossover
Selection
3
4
Vi
Vi+1
5
6
Profileri
Profileri+1
7
9
Trace Statistics
8
Method Statistics
Execution Trace Analyzer
14
Stage 1
GA component
Changes
11
12
Impact Analysis
13
Impact Sets
16
Mining
17
Ranked lists of changes
15
Stage 2
CIA component



Initial Inputs
1
Inputs
2
JMeter
10
GA Analyzer
Mutation
Crossover
Selection
3
4
Vi
Vi+1
5
6
Profileri
Profileri+1
7
Execution Trace Analyzer
8
9
Trace Statistics
Method Statistics
14
15
Changes
11
12
Impact Analysis
13
Impact Sets
16
Mining
17
Ranked lists of changes
Stage 1
GA component
Stage 2
CIA component



Change Impact:

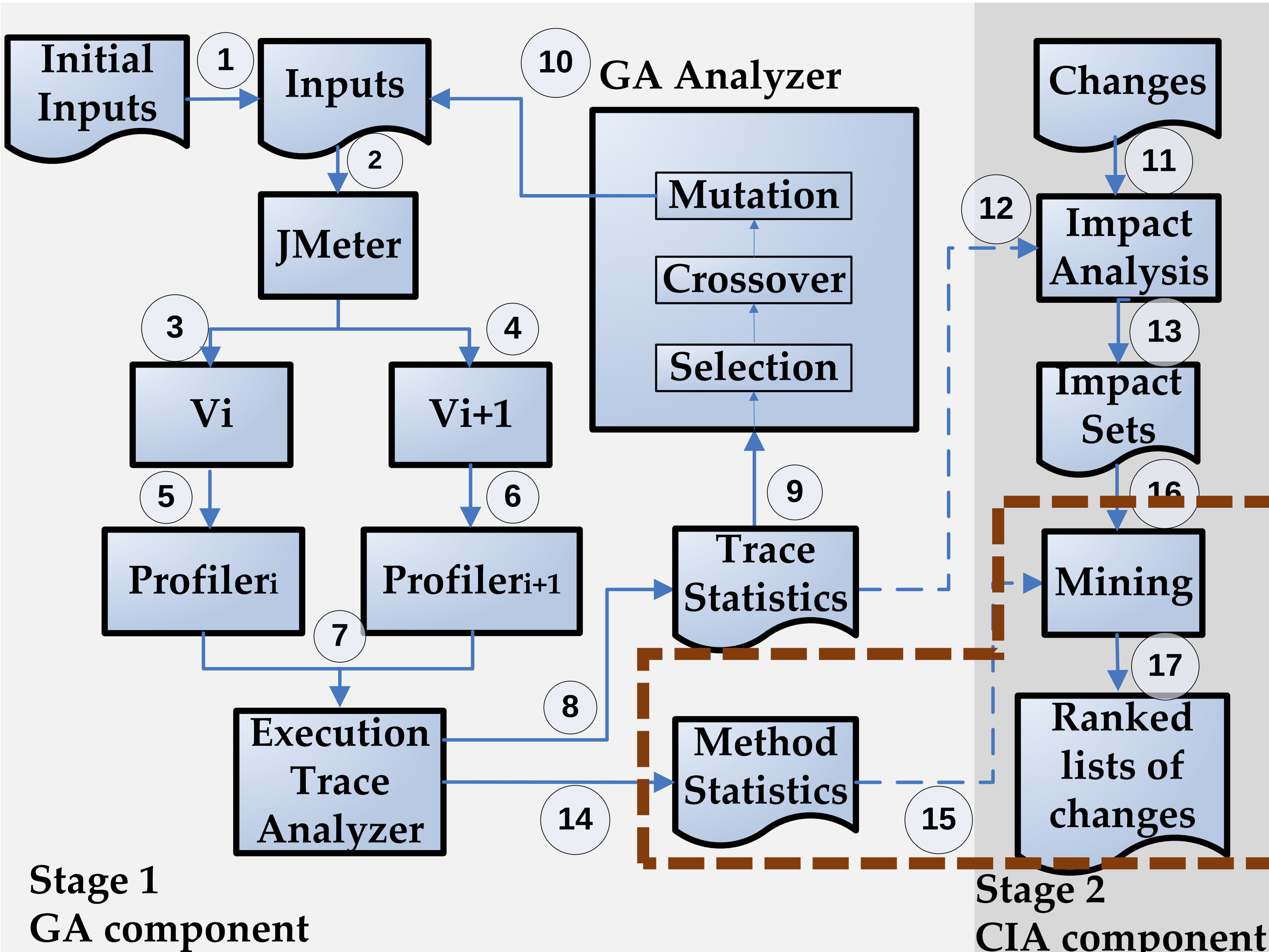
propagates along any (and only) dynamic paths that pass through the change (Law et al. ICSE'03)

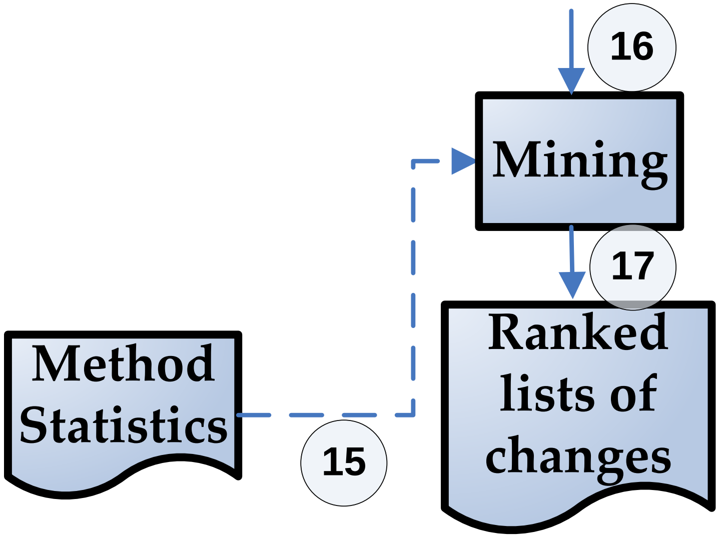
Impact Sets:

C_A => $M_A, M_D, M_Q, \ldots$

C_B => $M_B, M_D, M_N, \ldots.$

…


Initial Inputs
1
Inputs
2
JMeter
3
4
Vi
Vi+1
5
6
Profileri
Profileri+1
7
Execution Trace Analyzer
8
14
10
GA Analyzer
Mutation
Crossover
Selection
9
Trace Statistics
Method Statistics
15
Changes
11
12
Impact Analysis
13
Impact Sets
16
Mining
17
Ranked lists of changes
Stage 1
GA component
Stage 2
CIA component



Impact Sets:

12 6 91 ms

$C_A => M_A, M_D, M_Q, \ldots$

37 6 45 ms

$C_B => M_B, M_D, M_N, \ldots.$

$\ldots$

Ranked list:

- C_B
- C_A
- …

Research Questions (RQs)

- RQ1 - How effective is PerfImpact in finding inputs that likely expose performance regressions
- RQ2 – Can PerfImpact effectively recommend changes likely responsible for performance regressions

Research Question 1

How effective is PerfImpact in finding inputs that likely expose regressions

➡ PerfImpact vs. Random

H_0: There is no statistically significant difference between PerfImpact and Random

Research Question 2

How effective is PerfImpact in identifying regression-inducing code changes

- Inject nine artificial code changes
- Extract real code changes

Experimental Design

Agilefant

V3.2 vs V3.3

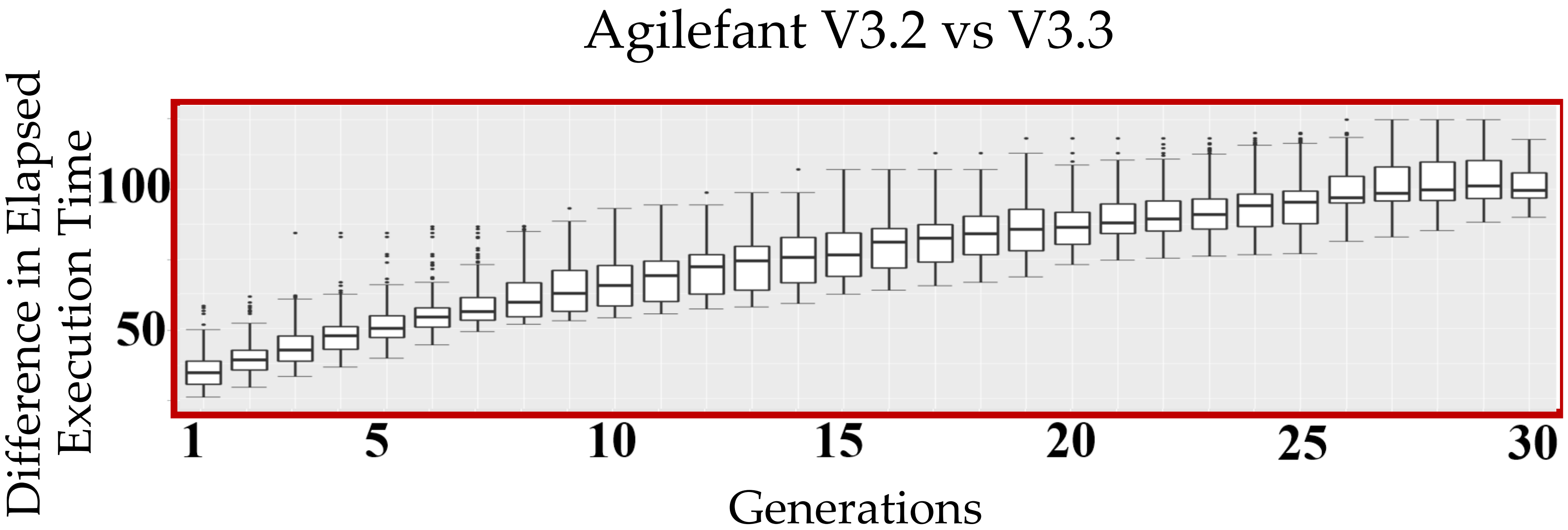
V3.2 vs V 3.5

JPetStore

V3.0.0 vs V4.0.5

Total: 187 real changes + 9 artificial changes

RQ1 – Finding Regression-Specific Inputs

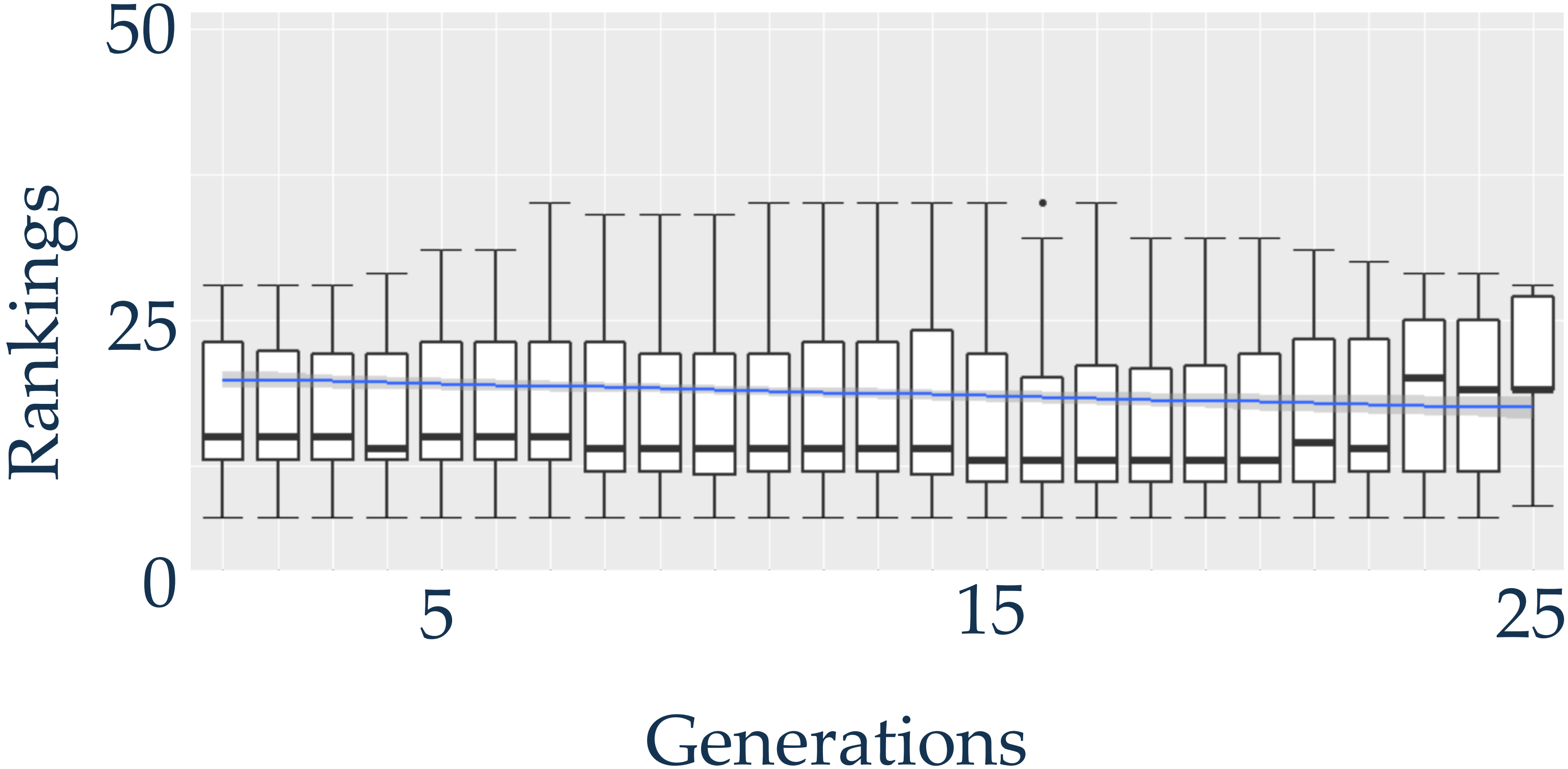


$p < 1.23e - 296$ => null hypothesis is rejected

RQ2 -Performance regression in a real code change

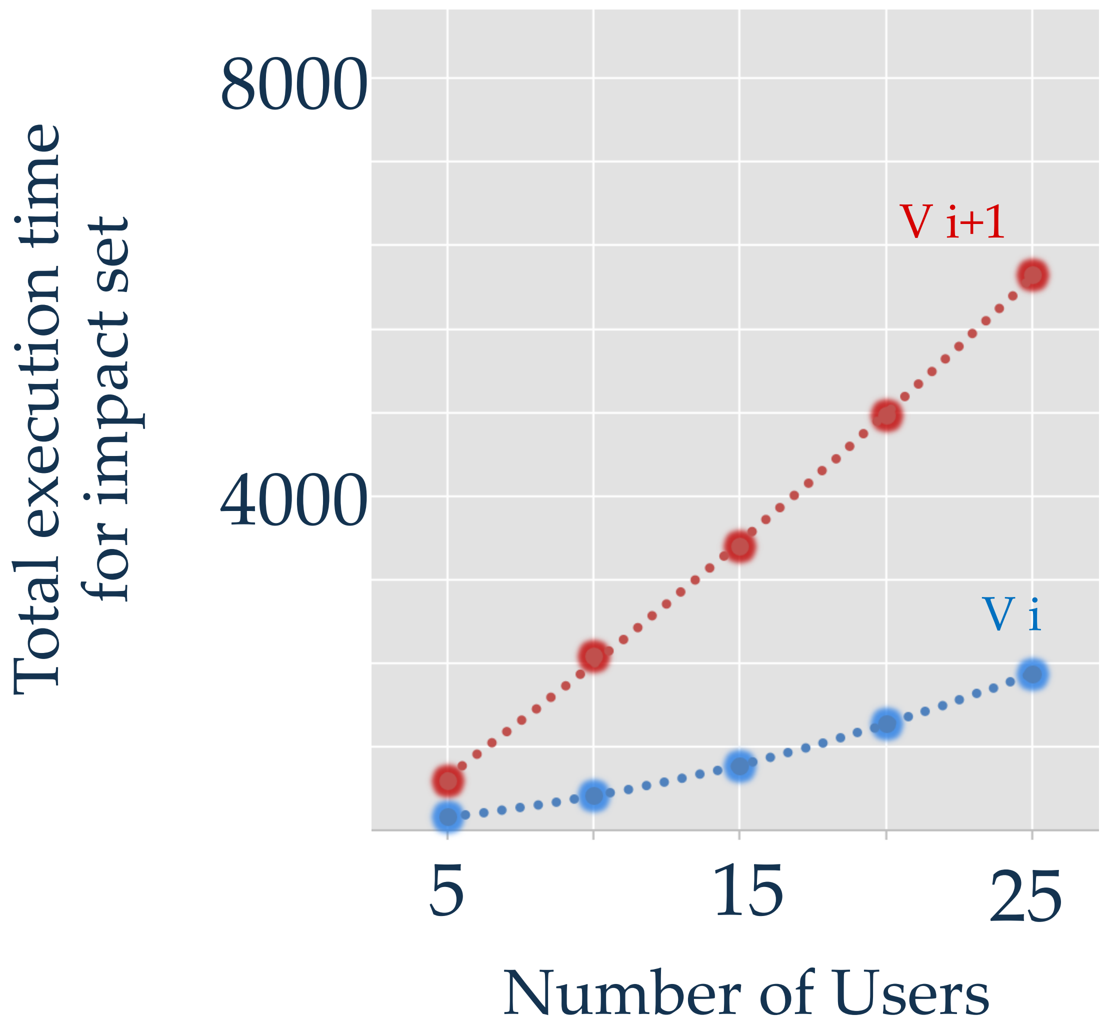
Code change: ProjectBusinessImpl.retrieveLeafStories()

Smaller values imply higher ranks



RQ2 -Performance regression in a real code change

Code change: ProjectBusinessImpl.retrieveLeafStories()



RQ2 -Performance regression in a real code change

Code change: ProjectBusinessImpl.retrieveLeafStories()

```
public List<StoryTO> retrieveLeafStories(int projectId, StoryFilters filters) {
      ......
      for(Story leafStory : leafStories) {
          StoryTO tmp = new StoryTO(leafStory); ......
          Set<Task> tasks = new HashSet<Task>();
          for (Task task : tmp.getTasks()) {
              TaskTO taskTO = new TaskTO(task);
              tasks.add(taskTO);
          }
          tmp.setTasks(tasks);
          ......
     }
     return leafStoriesWithRank;
}
```

Mining Performance Regression Inducing Code Changes in Evolving Software

Qi Luo
College of William and Mary
qluo@cs.wm.edu

Denys Poshyvanyk
College of William and Mary
denys@cs.wm.edu

Mark Grechanik
University of Illinois at Chicago
drmark@uic.edu

ABSTRACT

During software evolution, the source code of a system frequently changes due to bug fixes or new feature requests. Some of these changes may accidentally degrade performance of a newly released software version. A notable problem of regression testing is how to find problematic changes

1. INTRODUCTION

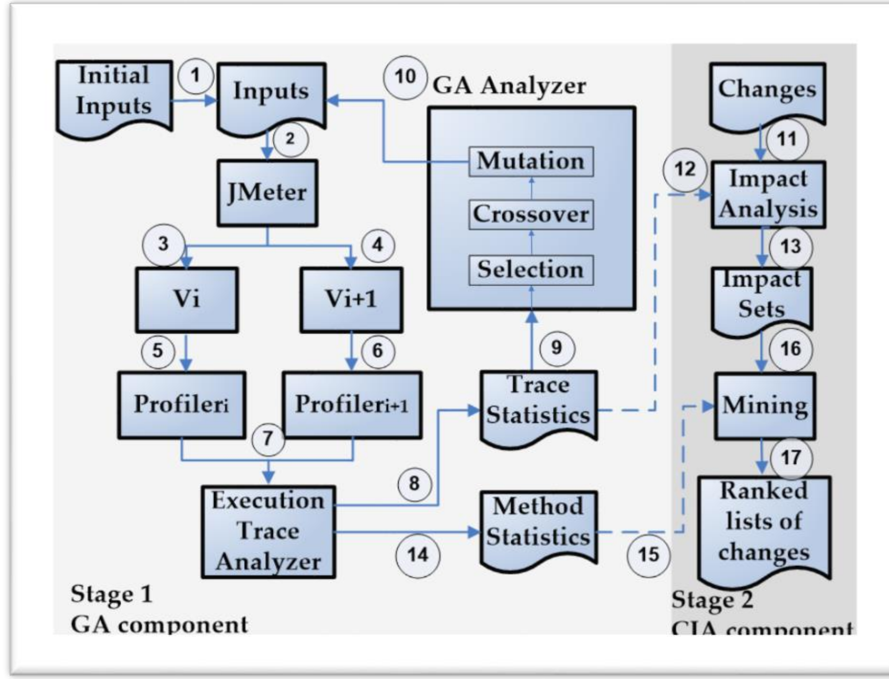
Performance is an important metric of software quality [60, 43], whereas performance testing is a vital activity that developers routinely perform during software development and maintenance to ensure quality [19]. During software evolution, a number of code changes are committed, and

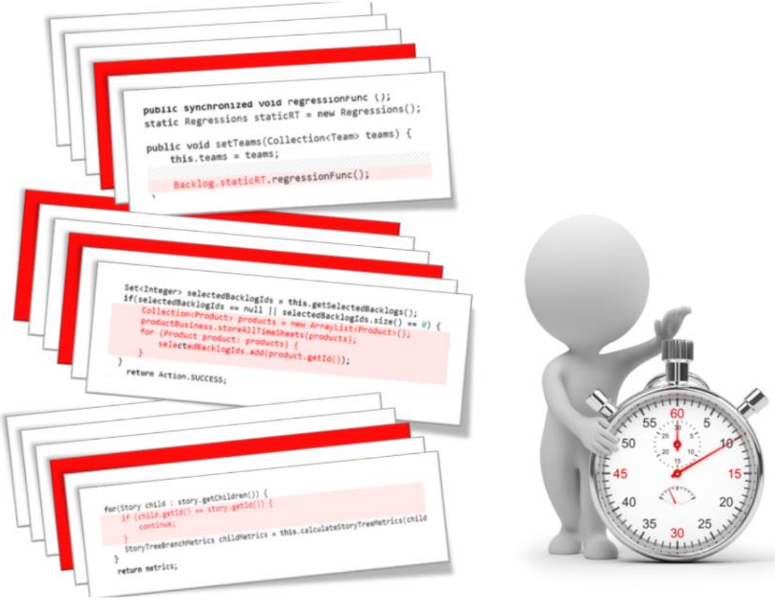
Online appendix:

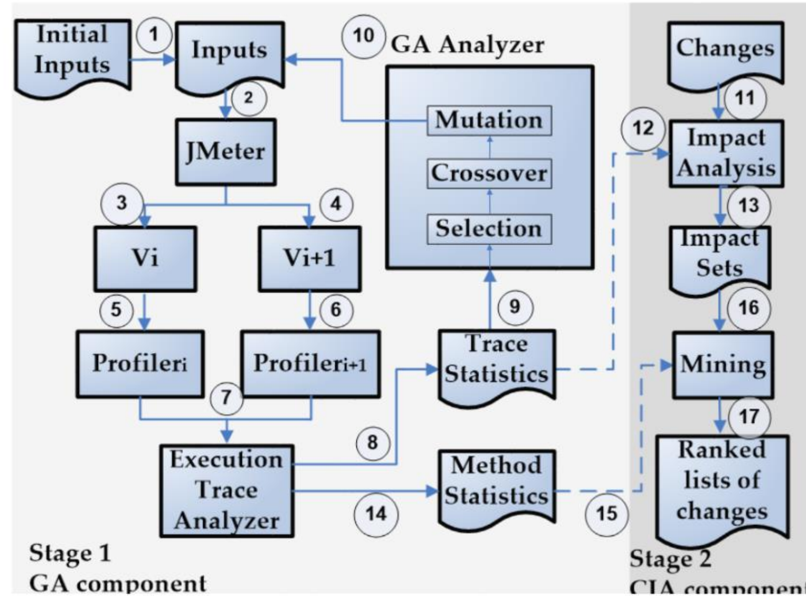
http://www.cs.wm.edu/semeru/data/MSR16-PerfImpact/

public synchronized void regressionFunc ();
static Regressions staticRT = new Regressions();
public void setTeams(Collection<Team> teams) {
this.teams = teams;
Backlog.staticRT.regressionFunc();
Set<Integer> selectedBacklogIds = this.getSelectedBacklogs();
if(selectedBacklogIds == null || selectedBacklogIds.size() == 0) {
Collection<Product> products = new ArrayList<Product>();
productBusiness.storeAllTimeSheets(products);
for (Product product: products) {
selectedBacklogIds.add(product.getId());
}
}
return Action.SUCCESS;
for(Story child : story.getChildren()) {
if (child.getId() == story.getId()) {
continue;
}
StoryTreeBranchMetrics childMetrics = this.calculateStoryTreeMetrics(child
}
return metrics;


public synchronized void regressionFunc ();
static Regressions staticRT = new Regressions();
public void setTeams(Collection<Team> teams) {
this.teams = teams;
Backlog.staticRT.regressionFunc();
return Action.SUCCESS;
return metrics;


Initial Inputs
1
Inputs
2
10
GA Analyzer
Mutation
Crossover
Selection
Changes
11
12
Impact Analysis
13
JMeter
3
4
Vi
Vi+1
Impact Sets
5
6
9
16
Profileri
Profileri+1
Trace Statistics
Mining
7
17
8
Execution Trace Analyzer
Method Statistics
Ranked lists of changes
14
15
Stage 1
GA component
Stage 2
CIA component





Experimental Design

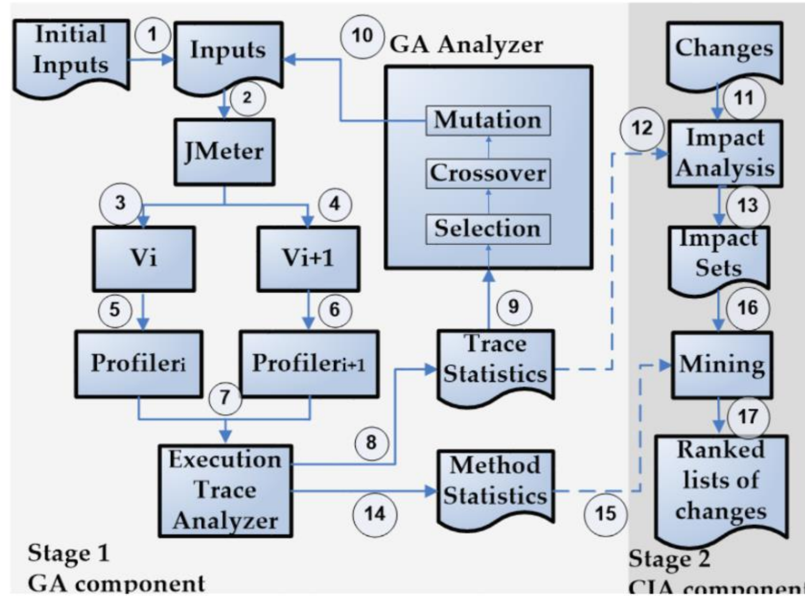
Agilefant

V3.2 vs V3.3
V3.2 vs V 3.5

JPetStore

V3.0.0 vs V4.0.5

'otal: 187 real changes + 9 artificial changes



Experimental Design

Agilefant

V3.2 vs V3.3
V3.2 vs V 3.5

JPetStore

V3.0.0 vs V4.0.5

'otal: 187 real changes + 9 artificial changes

PerfImpact

- PerfImpact is effective in finding specific inputs that expose performance regressions
- PerfImpact is effective in identifying regression-inducing code changes

Additional Slides for Questions

GAs – Independent Variables

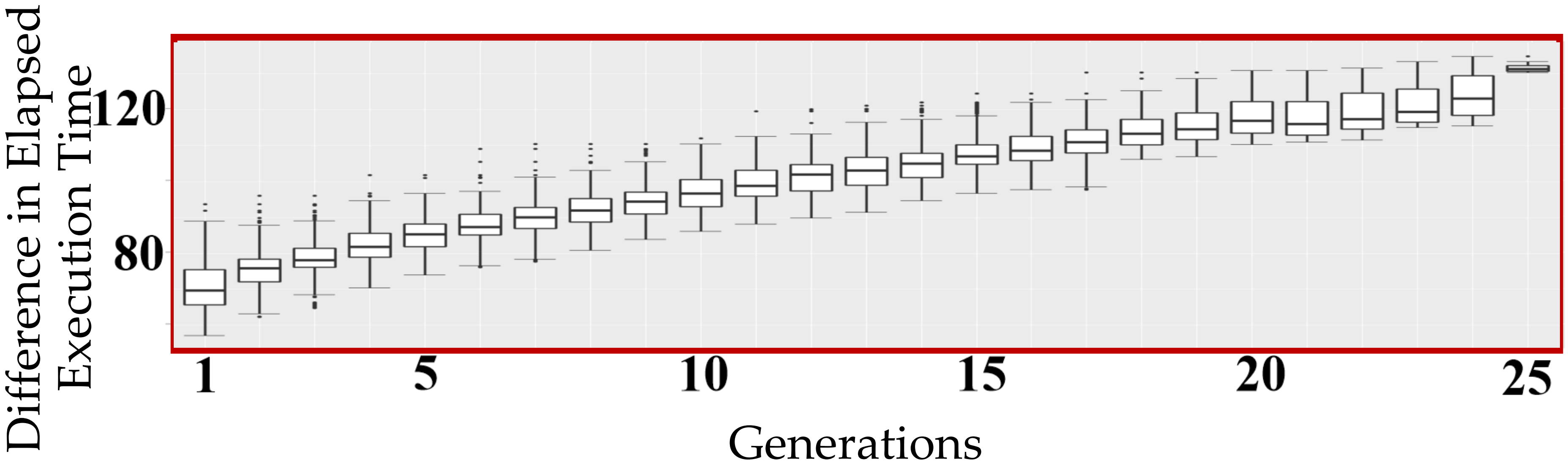
- Crossover rate – 0.3
- Mutation rate – 0.1
- Number of individuals per generation – 30
- Termination Criterion

  Maximum limit for the number of generations – 30

  Average fitness value of every individual in one generation

RQ1 – Finding Regression-Specific Inputs

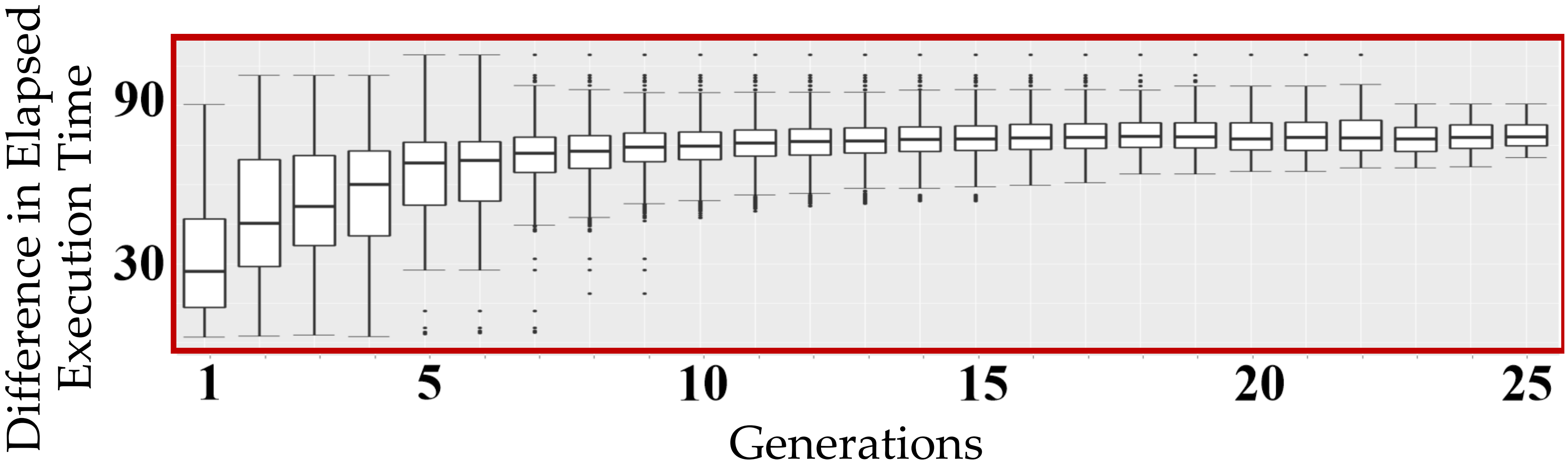
Agilefant V3.2 vs V3.5



$\boldsymbol{p = 1.37e - 236}$ => null hypothesis is rejected

RQ1 – Finding Regression-Specific Inputs

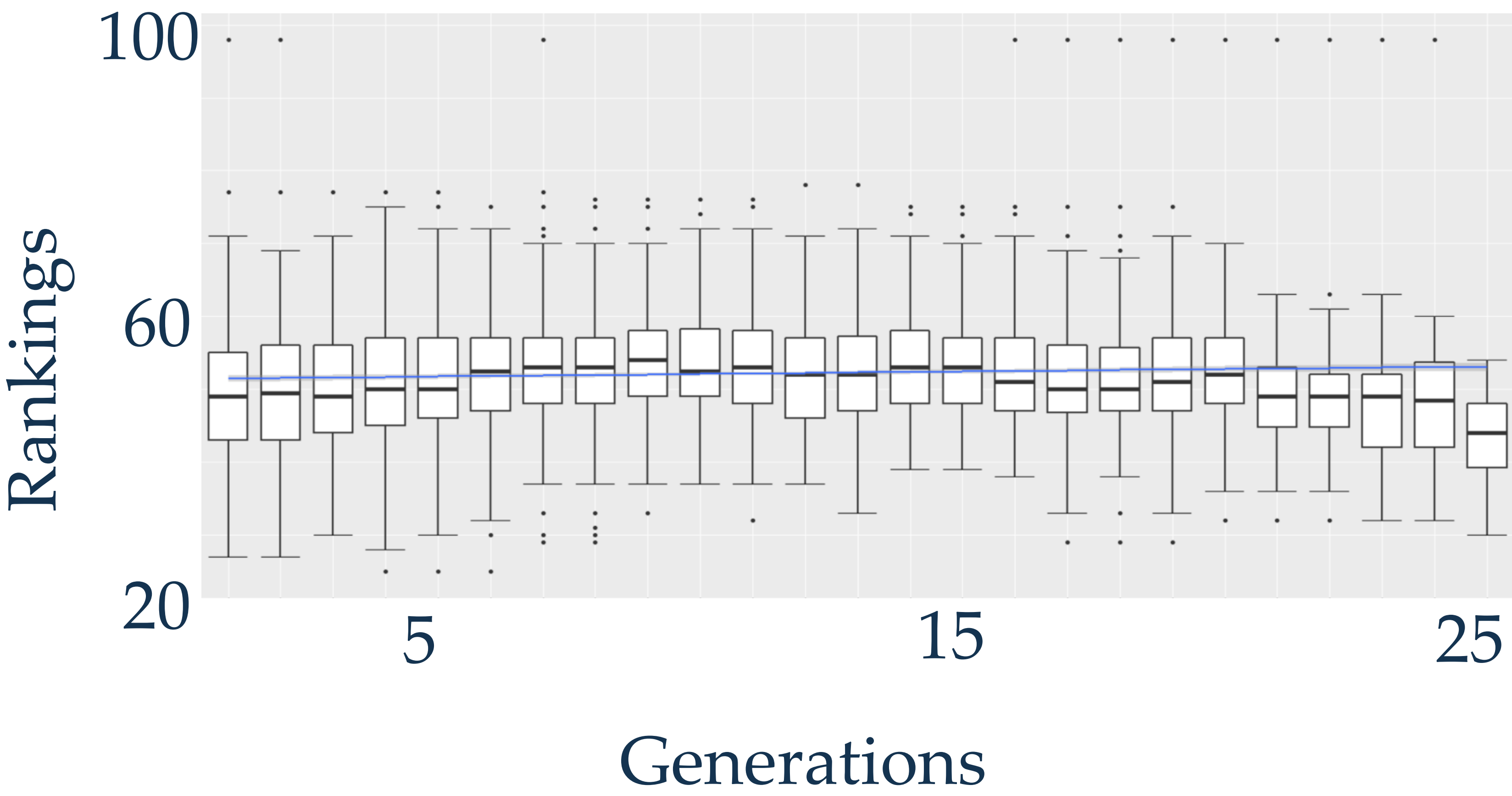
JPetStore V3.0.0 vs V4.0.5



$p = 2.64e - 198$ => null hypothesis is rejected

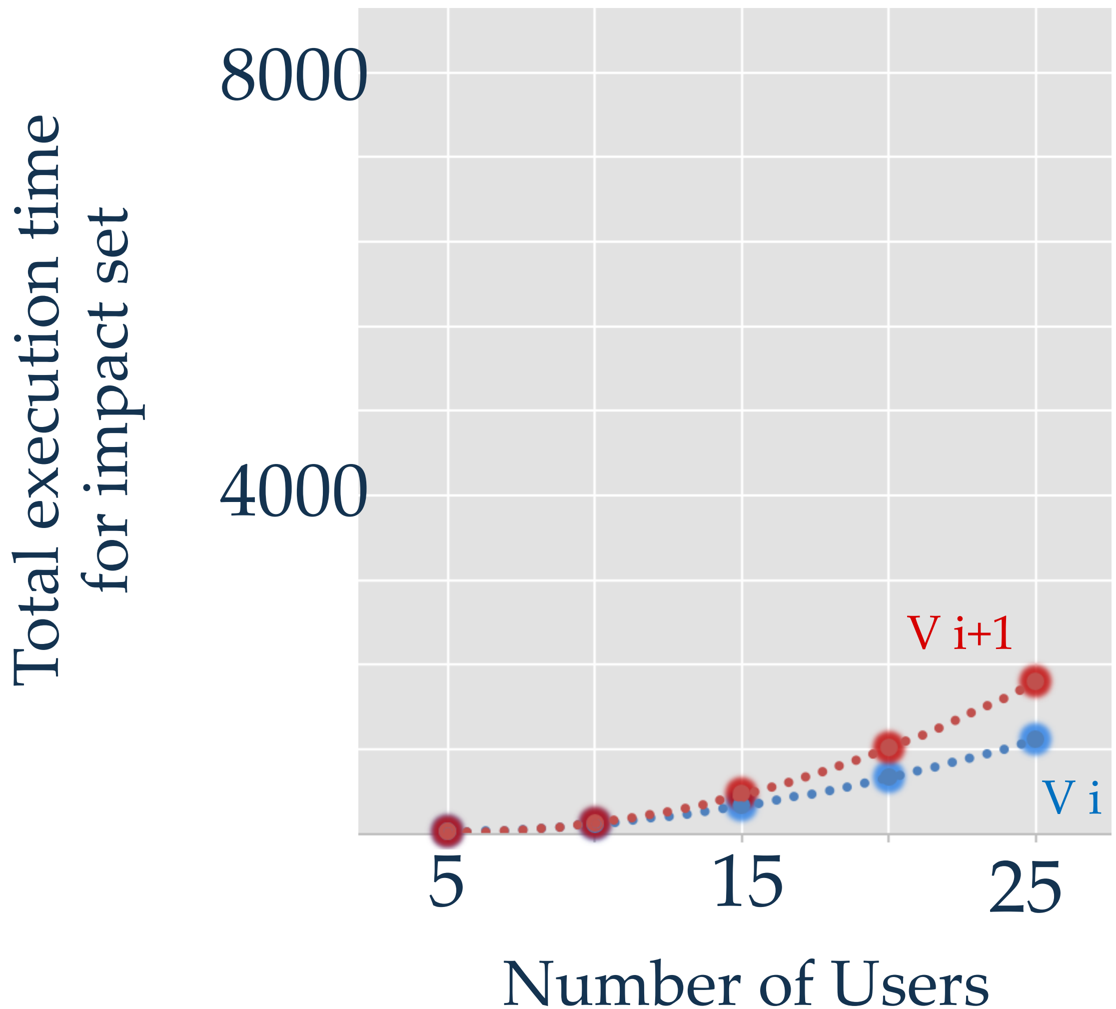
RQ$_2$ -Performance regression in a real code change

Code change: StoryHierarchyBusinessImpl.calculateStoryTreeMetrics()



RQ2 -Performance regression in a real code change

Code change: StoryHierarchyBusinessImpl.calculateStoryTreeMetrics()

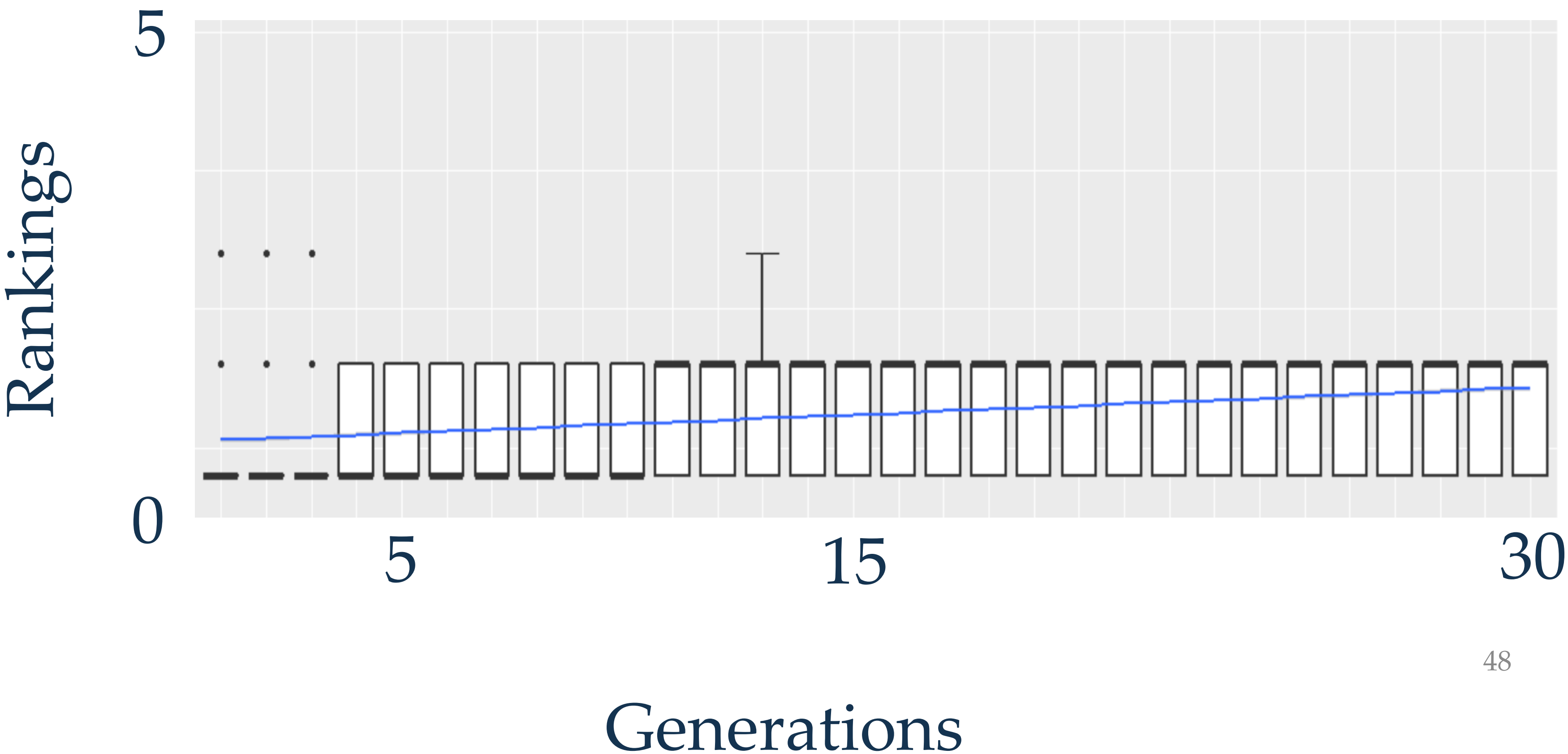


RQ2 -Performance regression in a real code change

Code change: StoryHierarchyBusinessImpl.calculateStoryTreeMetrics()

```
public StoryTreeBranchMetrics calculateStoryTreeMetrics(Story story) {
  for(Story child : story.getChildren()) {
      if (child.getId() == story.getId()) {
          continue;
      }
      StoryTreeBranchMetrics childMetrics = this.calculateStoryTreeMetrics(child);
   }
   return metrics;
}
```

RQ2 -Performance regression in an artificial code change



RQ2 -Performance regression in an artificial code change