REGULAR PAPER



Bridging the model-to-code abstraction gap with fuzzy logic in model-based regression test selection

Walter Cazzola¹ → Sudipto Ghosh² → Mohammed Al-Refai³ · Gabriele Maurina¹

Received: 14 September 2020 / Revised: 19 March 2021 / Accepted: 1 June 2021 / Published online: 21 July 2021 © The Author(s) 2021

Abstract

Regression test selection (RTS) approaches reduce the cost of regression testing of evolving software systems. Existing RTS approaches based on UML models use behavioral diagrams or a combination of structural and behavioral diagrams. However, in practice, behavioral diagrams are incomplete or not used. In previous work, we proposed a fuzzy logic based RTS approach called FLiRTS that uses UML sequence and activity diagrams. In this work, we introduce FLiRTS 2, which drops the need for behavioral diagrams and relies on system models that only use UML class diagrams, which are the most widely used UML diagrams in practice. FLiRTS 2 addresses the unavailability of behavioral diagrams by classifying test cases using fuzzy logic after analyzing the information commonly provided in class diagrams. We evaluated FLiRTS 2 on UML class diagrams extracted from 3331 revisions of 13 open-source software systems, and compared the results with those of code-based dynamic (Ekstazi) and static (STARTS) RTS approaches. The average test suite reduction using FLiRTS 2 was 82.06%. The average safety violations of FLiRTS 2 with respect to Ekstazi and STARTS were 18.88% and 16.53%, respectively. FLiRTS 2 selected on average about 82% of the test cases that were selected by Ekstazi and STARTS. The average precision violations of FLiRTS 2 with respect to Ekstazi and STARTS were 13.27% and 9.01%, respectively. The average mutation score of the full test suites was 18.90%; the standard deviation of the reduced test suites from the average deviation of the mutation score for each subject was 1.78% for FLiRTS 2, 1.11% for Ekstazi, and 1.43% for STARTS. Our experiment demonstrated that the performance of FLiRTS 2 is close to the state-of-art tools for code-based RTS but requires less information and performs the selection in less time.

Keywords Class diagram · Fuzzy logic · Regression test selection · UML

Communicated by Dimitris Kolovos.

Walter Cazzola cazzola@di.unimi.it https://cazzola.di.unimi.it

Sudipto Ghosh ghosh@cs.colostate.edu

Mohammed Al-Refai mnalrefai@just.edu.jo

Gabriele Maurina gabriele.maurina@studenti.unimi.it

- Computer Science Department, Università degli Studi di Milano, Milan, Italy
- ² Computer Science Department, Colorado State University, Fort Collins, USA
- ³ Computer Science Department, Jordan University of Science and Technology, Irbid, Jordan

1 Introduction

Regression testing is an expensive activity that ensures that changes made to evolving software systems do not break previously tested functionality [32,61]. Regression test selection (RTS) approaches improve regression testing efficiency by executing a subset of the original test suite instead of the full test suite [11,32]. RTS is performed by analyzing the changes made to software at the code [32] or model level [11]. Leung and White [47] show that a selective-retest technique becomes beneficial when the cost of test selection is less than the cost of running the non-selected tests. This makes it evident that reducing the cost of the selection process is important.

Model-based RTS offers several advantages over codebased RTS [11,76], such as higher scalability [76] and also reusability because it exploits widely used modeling notations (e.g., UML) that are independent of the programming



language used to implement the system [11]. Briand et al. [11] state that early estimation of regression testing effort is possible after modifying the design of the system and before propagating the changes to the code [11].

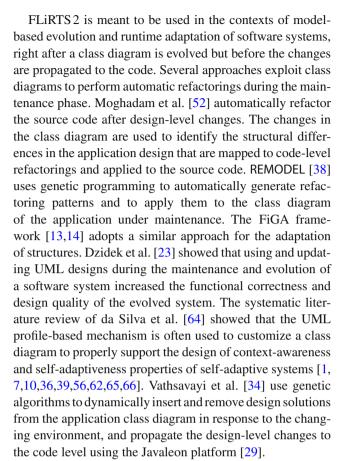
The existing UML model-based approaches use behavioral diagrams [15,40,75] or a combination of structural and behavioral diagrams [3,4,11,25,79]. Behavioral diagrams (e.g., activity, state, and sequence diagrams) represent calls between system operations, control dependencies, and object interactions. This information is needed to perform impact analysis on the changed model elements and to build traceability links between model elements and test cases.

Table 1 summarizes the results of evaluation of opensource projects [54] and developer surveys [21,22,26,30,37, 55,72] to assess the use of UML diagrams in real-world software development. The percentages were calculated as the ratio of the number of respondents (or projects) that use a specific diagram type to the total number of the respondents (or projects). The values in a column do not add up to 100% because a respondent (or a project) may use multiple diagram types. Behavioral diagrams are clearly used less often than class diagrams, which limits the applicability of existing model-based RTS approaches. The difference in the usage frequency between the class diagram and the three diagram types used in model-based RTS approaches (activity, sequence, and state diagrams) is more than 20% in 5 studies. In the other 3 studies, the difference ranges between 4 and 52% depending on the diagram type.

Even when both types of diagrams are used, the behavioral diagrams may be inconsistent with the structural diagrams or incomplete. Lange et al. [42]'s analysis of 14 industrial UML models of different sizes from various organizations found that between 40 and 78% of the operations represented in the class diagrams were not called in the sequence diagrams. Between 35 and 61% of the classes represented in the class diagrams did not occur as objects in the sequence diagrams.

For the above reasons, it is desirable to develop an RTS approach that *only* uses class diagrams. In [2], we proposed a model-based RTS approach called FLiRTS. It uses fuzzy logic to fill the abstraction gap of high-level (and therefore, incomplete) sequence and activity diagrams during test selection. In this paper, we propose FLiRTS 2, which extends FLiRTS for wider applicability by dropping the need for behavioral diagrams and focusing solely on UML class diagrams.

FLiRTS 2 requires as inputs (1) a class diagram modeling the software system and its test classes, and (2) the names of classes that changed between the two versions. Langer et al. [43] showed that class diagrams used in practice contain classes and interfaces, operation signatures and return types, generalization and realization relationships, and associations.



We compared the safety and precision violation, test selection reduction, and fault-detection ability of FLiRTS 2 with those of a dynamic RTS approach (Ekstazi [27]) and a static RTS approach (STARTS [45]). We used UML class diagrams extracted from 3331 revisions of 13 open-source software systems.

This paper is organized as follows. Sect. 2 provides background on fuzzy logic and presents the FLiRTS 2 approach and its tuning process. Sect. 3 describes the evaluation. Related work is summarized in Sect. 4. Conclusions and plans for future work are outlined in Sect. 5.

2 FLiRTS 2 approach

This section provides background on fuzzy logic (Sect. 2.1), an overview of FLiRTS 2 (Sect. 2.2), a description of its main steps (Sect. 2.3), and the process used to tune the fuzzy logic system and FLiRTS 2 parameters (Sect. 2.4).

2.1 Fuzzy logic

Fuzzy logic provides reasoning mechanisms to deal with uncertainty. It is a many-valued logic in which the truth values are not limited to *true* and *false* but may range between *completely true* and *completely false* [78]. A fuzzy logic system is



Table 1 Percentage of UML diagram type usage

UML diagram	[21,22] (%)	[30] (%)	[72] (%)	[37] (%)	[26] (%)	[55] (%)	[54] (%)
Class	73	93	99	85	61	63	100
Activity	32	60	47	56	33	54	20
Sequence	50	89	94	35	41	54	20
State machine	29	63	91	22	8	27	0

a control system that is based on fuzzy logic and is realized by three steps: fuzzification, inference, and defuzzification [8]. The inputs to a fuzzy logic system are called input crisp variables, which take discrete values called input crisp values. The fuzzification step maps the input crisp values to input fuzzy values by using input fuzzy sets. A fuzzy set is one that allows its members to have different values of membership in the interval [0,1] based on a defined membership function. The inference step evaluates predefined *inference rules* using the input fuzzy values. The rules are formulated in the form "if antecedent then consequent", where the antecedent can be any number of logical statements. An output variable and its fuzzy sets are also defined, and they are represented in the consequents of the inference rules. After evaluating the inference rules, their consequents are combined to form an aggregated fuzzy set from the output fuzzy sets. The defuzzification step produces an output crisp value from the aggregated fuzzy set. The *output crisp value* is mapped through the membership functions of the output fuzzy sets to a value between 0 and 1. If the value exceeds a predefined threshold, then the output set that produced the value becomes the final decision of the fuzzy logic system.

2.2 FLiRTS 2 overview

FLiRTS 2 classifies test cases as retestable or reusable based on the changes made to the UML class diagram representing the system under test. Retestable test cases exercise the modified parts of the system and must be re-executed. Reusable test cases only exercise the unmodified parts of the system and can be discarded [76]. Consistent with the current trend in code-based RTS research [27,45,81], FLiRTS 2 considers a test class to be a test case. It supports both unit and system test cases.

FLiRTS 2 depends on the knowledge of which classes under test are directly invoked by each test class. These invocations are modeled in the class diagram as *call usage dependency* relationships [9]. The class diagram must contain the test classes, their *call usage dependencies*, and the classes under test. No other *usage dependency* relationships in the class diagram, such as the *create* and *send* dependencies, or usage dependencies with customized stereotypes are required. However, if provided, they are treated as *call usage dependencies*.

FLiRTS 2 can be used with class diagrams that are customized using the UML-profile mechanism for a particular domain. A profile is defined using stereotypes, their tagged values, and constraints, which are applied to specific model elements, such as classes and operations. FLiRTS 2 supports stereotypes, tagged values, and constraints, but they are optional.

FLiRTS 2 does not need any behavioral diagram nor static/dynamic call dependency information except for the one from the test classes. From the UML class diagram, FLiRTS 2 constructs a graph, called class relationships graph (CRG), connecting the nodes representing the classes through edges representing the various relationships between them (e.g., associations and generalizations). From this CRG, it identifies all the paths from the test classes to the adapted classes. These paths are used as a substitute for the traceability information used in the canonical RTS approaches to classify test classes. The class diagram cannot provide complete information about the real usage of the diagram elements and whether or not the identified paths are actually exercised during the testing. The class diagram provides a static view of the application and the paths are identified with varying degrees of confidence that the relationships are actually exercised during the testing. This leads to an abstraction gap between the static model and the actual execution. FLiRTS 2 addresses this gap by adopting a probabilistic approach based on fuzzy logic.

The input crisp variables are based on the types of the relationships between the classes. The variables model (1) the probability that the execution of a test class traverses some adapted classes, and (2) the minimum distance from a test class to an adapted class. The input crisp values are calculated for each test class and used by the fuzzy logic system to compute the probabilities of a test class belonging to the reusable and retestable output fuzzy sets. If the probability of a test class for being retestable is above a given threshold, then the test class is classified as reusable. Otherwise, the test class is classified as reusable.

The results of a fuzzy logic system depend on several factors that need to be carefully configured to get acceptable results. However, there are no general rules or guidelines for configuring these factors that are appropriate for every domain [17,19]. We fine-tuned the fuzzy logic system through a controlled experiment that selected the



best configuration by comparing the test classification results obtained by systematically varying the factors through several possible values.

2.3 FLiRTS 2 process

The process consists of five steps that are all automated:

- 1. Building the CRG from the class diagram (Sect. 2.3.1).
- 2. Marking adapted classes in the CRG (Sect. 2.3.2).
- 3. Calculating the paths to the adapted classes (Sect. 2.3.3).
- 4. Calculating the input crisp values (Sect. 2.3.4).
- 5. Classifying the test cases (Sect. 2.3.5).

We demonstrate these steps on a small example taken from Kapitsaki and Venieris [39] for a context-aware lecture reservation service provided in a university campus. The service is available to every faculty and visiting professor who wants to book a lecture room. The context-aware service shows room availability only for those campus buildings close to the user's current location. The list of available rooms is based on weather conditions because the campus provides a number of open air facilities for lectures.

In the original design, the weather—implemented by the class Weather—could only assume the states *sunny*, *rain*, or *snow* without any relationship to the temperature. In the adapted design [39] shown in Fig. 1, the temperature concept is added for use with the *good weather* condition. The added elements were the classes Temperature and DigitalThermometer, an aggregation relationship from Weather to Temperature, a usage dependency relationship from Temperature to DigitalThermometer, and the necessary stereotypes and tag values.

The class diagram did not include test classes. We added two test classes LectureReservationAppTest and LocationProviderAppTest, and these are shown in Fig. 1.

2.3.1 Building the CRG from the class diagram

The CRG is a weighted directed graph extracted from the adapted class diagram. We need to capture the directions of the element types because a directed edge from node A to node B indicates a likelihood that the element represented by A calls operations in the element represented by B.

Each class/interface is represented as a node in the CRG. Weighted directed edges are added between the nodes in the CRG when the corresponding classes/interfaces in the class diagram are connected by one of the UML element types: association, generalization, realization, formal parameter and return types of operations, stereotype tagged values, and usage dependency relationships. Figure 2 shows how these

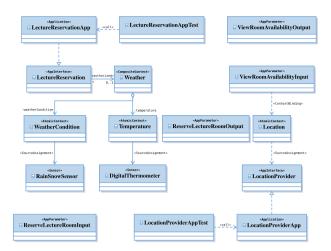


Fig. 1 Partial class diagram after adaptation [39]

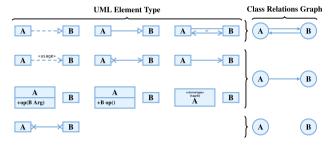


Fig. 2 Mapping rules from UML element types to CRG

element types are mapped into directed edges in the CRG. OCL expressions may be used in the class diagram. However, no extra information is obtained regarding the relationships because the navigation between classes in the OCL expressions uses the associations and operation parameters that we already use to add edges to the CRG.

Each element type implicitly has a different likelihood to be exercised by a test execution and therefore to drive the execution of some adapted class. Such a likelihood depends on several factors that are unpredictable. Each edge in the CRG has a weight that represents the likelihood. Each weight is 2 to a power from 0 to 6. For the associations shown in the first three rows of Fig. 2, we use the same weight but the association type determines the number and direction of the edges that are added to the CRG. We also do not assign different weights to associations, aggregation, and composition relationships. Similarly, all types of *usage dependency* relationships are assigned an equal weight. The actual weight assigned to each element type is determined by the tuning process described in Sect. 2.4.

Only one directed edge e can be added from a node A to a node B in the CRG. The weight of the edge is:

$$\omega_e = \sum_{i=1}^N \omega_i$$



where N is the number of the UML element types introducing a relationship from class A to class B, and ω_i is the weight assigned to the UML element type. For example, suppose that there are 3 associations and 1 generalization from class A to class B and that the weights 8 and 16 are assigned to the associations and generalizations, respectively. Then, the weights of the edges from A to B and B to A are 40 (=8*3 + 16*1) and 16 (=16*1), respectively.

2.3.2 Marking adapted classes in the CRG

This step requires the names of the adapted classes for labeling the corresponding CRG nodes as adapted. For example, in Fig. 3, the nodes DigitalThermometer, Weather, and Temperature have a red border because they are changed and marked as adapted.

The names of the adapted classes can be obtained in multiple ways. For example, model comparison techniques [12, 51,73] can identify the adapted model elements and the classes/interfaces containing these elements by comparing the original with the adapted class diagram. Or they can be recorded during the adaptation of the original diagram by the used tool; this is the approach followed by the FiGA framework [13,14]. Briand et al. [11] suggest another approach where developers use stereotypes to indicate all the classes that they expect to be adapted.

2.3.3 Calculating paths to the adapted classes

The input crisp values in FLiRTS 2 are computed wrt. each test class t. We consider the acyclic subgraph $G_t = \{V_t, E_t\}$ of the CRG where V_t contains t and all the nodes of the CRG reachable from t and E_t contains all the edges composing the simple paths (i.e., paths that do not contain cycles) connecting the nodes of V_t in CRG. We also consider $G_t' = \{V_t', E_t'\}$ defined as the restriction of G_t over the adapted nodes reachable from t. Therefore, E_t' contains all the edges of the simple paths from t that reach an adapted node and V_t' contains the nodes composing the edges in E_t' . Note that both $V_t' \subseteq V_t$ and $E_t' \subseteq E_t$ by construction.

2.3.4 Calculating input crisp values

Two input crisp variables \mathbf{p} and \mathbf{d} are used. The probability that the execution of a test class traverses some adapted classes is represented by \mathbf{p} , which considers the weights assigned to the UML element types in the construction of the CRG. The shortest distance from a test class to an adapted class is represented by \mathbf{d} , where the shortest distance is the minimum number of edges in the CRG that must be exercised by a test class to reach an adapted class.

Given a test class t, and the graphs $G_t = \{V_t, E_t\}$ and $G'_t = \{V'_t, E'_t\}$ defined in Sect. 2.3.3, the value of \mathbf{p} wrt. t is

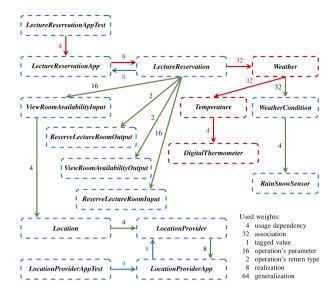


Fig. 3 Extracted CRG. Red nodes are the adapted classes. Red and blue edges are used to exemplify the calculation of the input crisp values

zero when V_t does not contain adapted nodes. Otherwise,

$$\mathbf{p} = \frac{\sum_{\varepsilon \in E_t'} \omega_{\varepsilon}}{\sum_{\eta \in E_t} \omega_{\eta}}$$

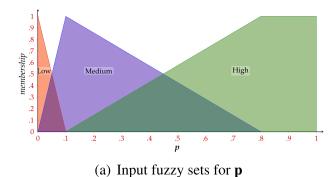
The higher the value of $\bf p$ for t, the higher the probability that the execution of t traverses some adapted classes. In the CRG of Fig. 3, only five edges (red colored) are traversed to reach an adapted node from the test class Lecture-ReservationAppTest; on the other hand, all the nodes are reachable from such test case and all but three edges (blue colored) can be traversed—the only excluded ones are those part of a cycle. The value of $\bf p$ for LectureReservationAppTest is 0.47 (i.e., (4+8+32+32+4)/(4+8+32+32+4+32+4+16+4+8+2+2+16)).

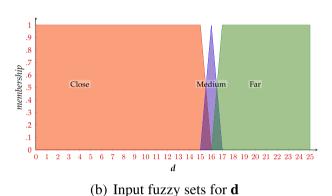
The value of \mathbf{d} wrt. t is infinity when no adapted node belongs to V_t . Otherwise, it is the number of edges in the shortest path connecting t to an adapted node in G_t' . From Fig. 3, the values of \mathbf{d} are 3 and infinity for LectureReservationAppTest and Location-ProviderAppTest, respectively. A smaller value of \mathbf{d} indicates that the test class must traverse fewer class relationships to reach some adapted class, and thus, the test class has a higher likelihood of being retestable.

2.3.5 Classifying the test cases

The classification process involves the *fuzzification*, *inference*, and *defuzzification* [8] phases of fuzzy-logic systems using the *input crisp variables* and *values* from Sect. 2.3.4.







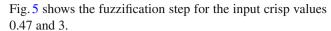
() I ... J

Fig. 4 Input fuzzy sets

Input fuzzy sets. For variable **p**, the sets are *Low*, *Medium*, and *High*, as shown in Fig. 4a. For variable **d**, the sets are *Close*, *Medium*, and *Far*, as shown in Fig. 4b. The membership functions of the input fuzzy sets of **p** and **d** were chosen according to the tuning process described in Sect. 2.4. The values assigned to **p** and **d** fit in these sets with a specific membership value between zero and one.

Output fuzzy sets. An output crisp variable, **tc**, is used for test classification. Its output fuzzy sets are *Retestable* and *Reusable* as shown in Fig. 5. These sets were defined to be of equal size and their membership functions are trapezoidal. The functions map the output crisp value to probabilities for the test class being retestable and reusable. The boundaries for the *Reusable* set are (0,1), (25,1) and (50,0); those for the *Retestable* set are (25,0), (60,1), and (75,1).

Fuzzification phase. This phase calculates the membership values in the input fuzzy sets for input crisp values assigned to **p** and **d**. The membership values are called fuzzy inputs, and they range between 0 and 1. The input crisp value of **p** for LectureReservationAppTest is 0.47. Using the fuzzy sets of **p** in Fig. 4a, the membership of 0.47 is zero in the Low set, 0.47 in the Medium set, and 0.53 in the High set. The input crisp value of **d** for LectureReservationAppTest is 3. Using the fuzzy sets of **d** in Fig. 4b, the membership of 3 is 1 in the Close set, zero in the Medium set, and zero in the Far set. Step (1) of



Inference phase. FLiRTS 2 uses nine inference rules to consider all the combinations for the fuzzy sets of $\bf p$ and $\bf d$. The antecedents of the rules are the input fuzzy sets of $\bf p$ and $\bf d$, and the consequents are the output fuzzy sets of $\bf tc$. The output value produced by each rule is called the *fuzzy output*. Table 2 summarizes the inference rules. The rules are in conjunctive canonical form where the conditions for the input crisp variables are connected by the AND (\wedge) operator. Figure 5 shows the application of 2 of the 9 rules.

The inference rules are evaluated using the Mamdani inference method [50], which is the most common inference method in practice and in the literature [60]. For each rule, the *fuzzy output* is calculated as the minimum of the *fuzzy input*s of the input crisp values used in the rule. For example, in Fig. 5, the input crisp values used in *rule 1* are 3 and 0.47, and their *fuzzy input*s are 1 ($\mu_{Close}(3) = 1$) and 0.47 ($\mu_{Medium}(0.47) = 0.47$), respectively. *Rule 1* is evaluated by applying the minimum operation between 1 and 0.47 and the result of *rule 1* is 0.47 as shown in step 2 of Fig. 5. Table 3 shows the results of applying each of the 9 rules.

The process of obtaining the overall consequent from the individual consequents of multiple rules is known as aggregation of rules (Step (3) of Fig. 5). We aggregate the outputs of all the rules using the maximum operation. From Table 3, the maximum output of all the rules that produce *Retestable* and *Reusable* is 0.53 and 0, respectively. The maximum outputs propagate through the *Retestable* and *Reusable* sets by truncating their respective membership functions. The truncated function of the *Retestable* set is shown as the shaded area in Fig. 5. The two truncated functions produce the aggregated membership function for the variable **tc**, shown as the output of Step (3) in Fig. 5.

Defuzzification phase. The final output crisp value is found using the center of gravity of the aggregated membership function for tc. This is the centroid method [16,60] and is the most common defuzzification method [60]. It finds the output crisp point x where a vertical line from x would slice the aggregated function into two equal masses. In our example, the defuzzification process returns 53.12 as the output crisp value as shown in Step (4) of Fig. 5. Finally, the output fuzzy sets, Retestable and Reusable, are used to map the output crisp value 53.12 to probabilities for Retestable and *Reusable*. The membership value of 53.12 in the *Retestable* set is 1 (as it can be spotted in Step (4) of Fig. 5 by crossing the prolonging of the slope of the Retestable set and the line perpendicular to the x-axis in 53.12), which means that the probability of LectureReservationAppTest being retestable is 100%.

The final classification of the test class is based on the probability of being *retestable* and a user-defined threshold. If the probability is above the threshold, the test class is classi-



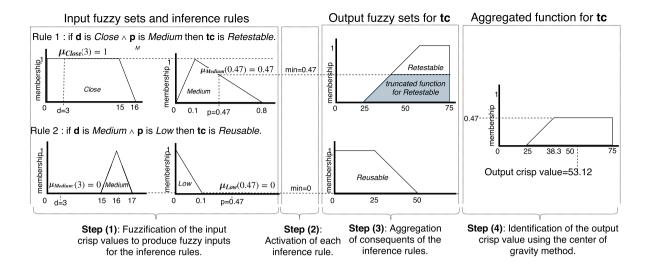


Fig. 5 Fuzzification, inference, and defuzzification phases

Table 2 Inference rules

p d	Close	Medium	Far
Low	Retestable	Reusable	Reusable
Medium	Retestable	Retestable	Retestable
High	Retestable	Retestable	Retestable

Table 3 Applying inference rules

p	$\mu_{Close}(3)=1$	$\mu_{Medium}(3)=0$	$\mu_{Far}(3)=0$
$\mu_{Low}(0.47)=0$	0	0	0
$\mu_{Medium}(0.47)=0.47$	0.47	0	0
$\mu_{High}(0.47)=0.53$	0.53	0	0

fied as retestable; otherwise, it is reusable. For example, if the threshold was 70%, then LectureReservationAppTest would be classified as retestable.

2.4 Tuning FLiRTS 2

As should be evident from Sect. 2.3, a *reliable* test case classification result relies on the use of a proper configuration of the fuzzy system. A configuration is a triplet of (1) a function that assigns a weight to each UML relationship type, (2) the input fuzzy sets for the variables \mathbf{p} and \mathbf{d} , and (3) the selection threshold to classify a test class as retestable.

The result provided by FLiRTS 2 can be considered *reliable* when the resulting test classification shows the lowest safety and precision violations and the highest test selection reduction [44,63] wrt. those achievable with other model-based or code-based RTS approaches on the same applications. In this respect, our term of comparison is

Ekstazi [27]. Ekstazi is a code-based RTS approach known to be safe in terms of selecting all the modification-traversing test classes, widely evaluated on a large number of revisions, and being adopted by several popular open source projects; as such it can be considered the state-of-the-art for RTS tools.

The tuning process consists of finding the configuration $\overline{c} \in C$ —the set of all possible configurations, or *configuration space*—that minimizes the safety and precision violations wrt. Ekstazi and maximizes the test selection reduction for a particular set of subjects S that we use as a *sample set*. Given $s \in S$, let R_s be the set of all considered revisions for s. Given $r \in R_s$, let E^{sr} and F_c^{sr} be the set of test cases of the revision r selected by Ekstazi and FLiRTS 2 with a configuration $c \in C$, respectively. Safety (SV_c^{sr}) and precision (PV_c^{sr}) violations and test selection reduction (TR_c^{sr}) for FLiRTS 2 with the configuration $c \in C$ wrt. Ekstazi on the subject $s \in S$ and revision $r \in R_s$ are calculated as:

$$SV_c^{sr} = \frac{|E^{sr} \smallsetminus F_c^{sr}|}{|E^{sr} \cup F_c^{sr}|} \quad PV_c^{sr} = \frac{|F_c^{sr} \smallsetminus E^{sr}|}{|E^{sr} \cup F_c^{sr}|} \quad TR_c^{sr} = \frac{|T^{sr}| - |F_c^{sr}|}{|T^{sr}|}$$

where T^{sr} is the test suite for the revision r of the subject s before RTS is performed. The values of SV_c^{sr} , PV_c^{sr} , and TR_c^{sr} are multiplied by 100 to make them percentages. Lower percentages for SV_c^{sr} and PV_c^{sr} , and higher percentages for TR_c^{sr} are better. SV_c^{sr} , PV_c^{sr} and TR_c^{sr} can be used to define, for each configuration c, the set of their average values over the revisions in $r \in R_S$ for each $s \in S$ as

$$V_c = \{A-SV_c^s, A-PV_c^s, A-TR_c^s | \forall s \in S\}$$

where

$$\begin{array}{l} A\text{-}SV_c^s = \text{average}\, SV_c^{sr} \\ r \in R_S \end{array}$$



$$\begin{aligned} A\text{-}PV_c^s &= \operatorname{average} PV_c^{sr} \\ r \in &R_S \end{aligned}$$

$$A\text{-}TR_c^s &= \operatorname{average} TR_c^{sr} \\ r \in &R_S \end{aligned}$$

They are also used to define for each subject $s \in S$, their optimum value over all the configurations in C as

$$opt = \{ < \min_{c \in C} A\text{-}SV^s_c, \min_{c \in C} A\text{-}PV^s_c, \max_{c \in C} A\text{-}TR^s_c > |\forall s \in S \}.$$

The best configuration $\overline{\mathbf{c}}$ is the one that shows the minimal distance from the optimum values for safety and precision violations and test selection reduction.

$$\exists \overline{c} \in C \text{ s.t.} \underset{c \in C}{\text{min Distance}}(opt, V_c)$$

where we used the Manhattan distance [20].

Section 2.4.1 presents the subject applications used in the tuning process. Sect. 2.4.2 describes the process of calculating the configuration space. We discuss issues about the tuning process and threats to validity in Sect. 2.4.3.

2.4.1 Subject applications for tuning FLiRTS 2

Even though FLiRTS 2 is a model-based approach, we are forced to tune it using code-based subjects because of the lack of large open-source model-based subjects and other model-based RTS tools. We used the 8 subjects listed in Table 5. These are selected from the 21 open-source Java projects summarized in Table 4, which are known to be compatible with Ekstazi since they were used in its evaluation [27,44].

We downloaded the revisions of every subject using the methodology in Legunsen et al. [44]. From the earliest revision to the latest **SHA** listed in Table 5, we chose all those revisions that (1) correctly compiled, and (2) for which the tests and Ekstazi compiled and ran successfully.

From source code to models. The UML class diagrams for the selected subjects were automatically extracted from each revision by using the Java to UML transformation plugin for the Rational Software Architect (RSA) [46]. The extracted models contain classes, interfaces, operations with input parameters and return types, associations, generalizations, and realizations. They do not contain usage dependency relationships because these are not supported by the RSA transformation plugin. Thus, the diagrams do not contain the information regarding the direct invocations from the test classes to the classes under test. We obtained this information from the corresponding code-level revision using Apache Commons BCEL [28], which supports static analysis of binary Java class files, and stored it in a text file. The text files were provided along with the corresponding class diagrams as inputs to FLiRTS 2. FLiRTS 2 supports reading the information regarding the direct invocations from

Table 4 Subjects for the tuning (plain) and the evaluation (shaded)

ID	Subject Name	ID	Subject Name
p_1	commons-net	p ₁₂	commons-dbcp
p_2	commons-collections	p ₁₃	commons-compress
p_3	commons-imaging	p ₁₄	commons-pool
p_4	asterisk-java	p ₁₅	commons-codec
p ₅	commons-jxpath	p ₁₆	commons-dbutils
p ₆	commons-configuration	p ₁₇	HikariCP
p ₇	commons-lang	p ₁₈	stream-lib
p ₈	commons-io	p ₁₉	commons-fileupload
p ₉	commons-cli	p ₂₀	commons-email
p ₁₀	commons-validator	p ₂₁	invokebinder
p ₁₁	commons-text		

the class diagram as well as from a separate text file when the information is not represented in the class diagram.

Marking adapted classes. As explained in Sect. 2.3, FLiRTS 2 needs the names of the adapted classes. However, EMF compare [12] and any other model comparison approaches do not work with the extracted models because each extracted model uses a different set of element *ids* and every comparison concludes that every model element has changed even if this is not the case. Thus, we were forced to use the git diff command on the source code. The names of the modified classes/interfaces produced by git diff were used as input to FLiRTS 2 to mark the adapted classes in the class diagram that was generated from the same source code.

Selecting revisions. Only revisions that differed from that previously selected by more than 3% of its Java classes were selected. The aim is to increase the chances of having multiple code changes between the selected revisions and to reduce the cases where the changes do not involve code modifications (e.g., only changes to comments).

We selected eight subjects (those with an empty background in Table 4): four with the smallest and four with the largest average number of changed classes. This choice permits us to tune FLiRTS 2 to work with both highly variable subjects (e.g., systems that introduce new functionality) and mostly stable subjects (e.g., systems whose changes are due to bug fixing). Table 5 shows for each subject, the latest revision used (\mathbf{SHA}), the number of used revisions (\mathbf{Revs}), and among all the used revisions, the average number of classes ($\mathbf{A_{cl}}$), and the average number of adapted classes ($\mathbf{A_{ad}}$).

2.4.2 Selecting the best configuration

The *configuration space* is the set of the combinations of values that can be assumed by the three elements in the triplet

< weight assignment, input fuzzy sets, selection threshold > .



Table 5 Selected projects for the tuning process

ID	SHA	A_{cl}	A_{ad}	Revs
p ₁	4e5a699	222	66	223
p_2	0dad342	510	58	41
p ₃	c873192	375	58	85
p_4	a3b01f4	471	43	163
P ₁₃	792da67	158	10	407
P ₁₅	db51a1c	85	7	318
P16	c66efa9	53	6	118
p_{20}	14d7643	38	4	122

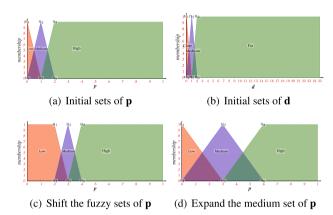


Fig. 6 Initial and generated input fuzzy sets

We now describe how these combinations were calculated.

Weight Assignment. This function assigns a weight among 1, 2, 4, 8, 16, and 32 to the considered UML element types: association, generalization, realization, formal parameters of operations, return types of operations, and usage dependency relationships. We did not consider tagged values because the class diagrams generated from the code-level revisions do not use UML profiles and do not contain tagged values. Because there are 6 weights for 6 element types, there are 720 (=6!) weight assignment functions.

Input fuzzy sets. The input fuzzy sets for the two input crisp variables \mathbf{p} and \mathbf{d} were generated by applying *shift* and *expand* operations to the initial sets. We only generated fuzzy sets with triangular or trapezoidal shapes because these layouts are proven to produce good results for most of the domains [17,19]. Fig. 6a, b shows the initial sets for \mathbf{p} and \mathbf{d} , and the points for the fuzzy sets from n_1 through n_6 . We apply the *shift* and *expand* operations to these points.

The input values range between 0 and 1 for \mathbf{p} , and 0 and 25 for \mathbf{d} . The range for \mathbf{d} was determined by considering all the values it could get from the test classes in the sample space \mathbf{S} and the largest value was 25, whereas \mathbf{p} is a probability and its values naturally range between 0 and 1. We applied *shift* and *expand* by a given amount ε , where ε was chosen

to be one-fifth of the range of \mathbf{p} (0–1) and \mathbf{d} (0–25). Thus, the value of ε was 0.2 for \mathbf{p} and 5 for \mathbf{d} .

The generation process starts from the initial values and repeatedly applies the shift operation using the amount ε to the right on all sets. To shift by ε means to add ε units to each point from n_1 to n_6 in each set. We stop shifting when the Medium set reaches the right boundary. This happens when the points on the right, n_3 and n_6 , are clipped at the right boundary. Figure 6c shows how the initial fuzzy sets of p were shifted. Each application of the shift operation generates a possible group of fuzzy sets. Each group is further refined by repeatedly applying an expand operation to the Medium set. Specifically, the point n_4 of the *Medium* set is moved to the left by ε , and the point n_6 of the *Medium* set is moved to the right by ε . The other sets are modified similarly. We repeatedly apply the expand operation until the left points n_1 and n_4 reach the left boundary and the right points n_3 and n_6 reach the right boundary. Figure 6b shows the result of an application of the expand operation to the Medium set of Fig. 6c. The expand operation (Fig. 6b) moved the four points $n_1, n_4, n_3, \text{ and } n_6.$

We generated 21 fuzzy set groups each for \mathbf{p} and \mathbf{d} . The fuzzy logic system uses one fuzzy set group for \mathbf{p} and one for \mathbf{d} together in a function block, which is a primitive object that contains the input and output variables and their fuzzy sets, and the inference rules [16]. Therefore, the total number of generated combinations was 21*21=441.

Selection threshold. We considered 5 possible values for the selection threshold: 50%, 60%, 70%, 80%, and 90%.

Configuration space. The total number of the configurations used to tune FLiRTS 2 was 720*441*5=1,587,600.

Best configuration. We ran Ekstazi on the selected subjects and FLiRTS 2 on the corresponding model inputs with each configuration from the calculated configuration space. The total number of FLiRTS 2 runs was 2,344,885,200 (=1,587,600 configurations * 1,477 revisions of the sample subjects). For every configuration, subject, and revision of a subject, we calculated the safety and precision violations, and test selection reduction wrt. Ekstazi and calculated the best configuration according to the formulas and process described in Sect. 2.4.

The best configuration uses the weights 2 for associations, 1 for realizations, 32 for generalizations, 4 for return types, 16 for input parameters, and 8 for usage dependencies. The input fuzzy sets for this configuration are shown in Fig. 4, and the selection threshold is 50%. This configuration is the one that shows the minimal distance wrt. the Ekstazi results for safety and precision violation and test selection reduction.

2.4.3 Threads to validity for the tuning process

FLiRTS 2 must be tuned only once for a new context and objective for which it is used. For example, we gave a higher



priority to safety than to test selection reduction. Practitioners who have different objectives, such as higher test selection reduction when regression testing time is limited as in continuous integration environments with frequent commits, will need to re-tune FLiRTS 2 to achieve higher test selection reduction. Re-tuning is also required if the approach is used for models where UML profiles and tagged values are used. In our tuning approach, we did not use models that have tagged values, and thus, we did not use weights for them.

Below we present threats to validity that can affect the outcome of the tuning process.

External validity. We used a small number (8) of sample subjects to tune FLiRTS 2, and used one criterion to select the subjects based on the largest/smallest number of changed classes. These sample projects may not be representative, so we cannot generalize the tuning result. Moreover, selecting the subjects based on other criteria such as the number and type of test cases could impact the tuning outcome. However, the selected subjects vary in size, application domain, number of revisions, and number of test classes, which reduces this threat.

Internal validity. We reverse engineered the class diagrams from code-level revisions. The generated associations were all directed, i.e., the class diagrams did not include the other types of associations shown in Fig. 2. Moreover, the generated diagrams did not include some design information such as tagged values of stereotypes and OCL expressions. Although providing this information is optional for FLiRTS 2, having it could impact the outcome.

We used the git diff command between the codelevel revisions to identify the adapted classes. Applying model comparison between the class diagrams could identify fewer adapted classes because some code-level changes may not be detectable at the model level. We extracted the direct invocations from test classes to classes under test from the code-level revisions. These deviations from FLiRTS 2, which should only use model-level information, could introduce errors and change the results.

The best configuration we found is a local optimum with respect to the configuration settings used in the tuning process. Using different weights, membership functions for the fuzzy sets, selection thresholds, and distance measures could impact the outcome.

Other factors that can affect the outcome are errors in the FLiRTS 2 implementation. To reduce this threat, we built our implementation on mature tools (i.e., JGraphT [53] and jFuzzyLogic [16]) and tested it thoroughly.

Construct validity. We chose Ekstazi as the ground truth against which to tune FLiRTS 2. However, Ekstazi may not represent the ground truth for all RTS, and tuning FLiRTS 2 with respect to other RTS approaches could impact the tuning outcome. However, Ekstazi is safe, and is similar to FLiRTS 2 in terms of supporting class-level RTS.

There are other techniques that can be used to tune fuzzy logic systems, such as genetic algorithms, which we did not consider in this work. We plan to investigate such techniques in the future.

3 Evaluation

The evaluation goals were to compare FLiRTS 2 with other RTS tools in terms of (1) safety violation, (2) precision violation, (3) test suite reduction, (4) selection performance and saved time for regression testing, and (5) fault-detection ability of the obtained reduced test suites. The terms safety violation, precision violation, and test suite reduction are defined in Sect. 2.4. The fault-detection ability was compared by using mutation analysis on the subjects using the full test suites and the reduced test suites obtained by the RTS tools.

To the best of our knowledge, there is no freely available repository that contains subjects with test cases and design models for several of their revisions. This forced us to extract the needed models from the source code of the considered subjects. It was not possible to compare FLiRTS 2 with other model-based RTS approaches because neither their tool implementations (e.g., [11,25]), nor the models used in the reported studies are available or because they (e.g., MaRTS [4]) use behavioral diagrams (e.g., sequence and activity diagrams) that are notoriously difficult to automatically extract from the source code. Therefore, we use two code-based RTS tools, Ekstazi [27] and STARTS [45]. They are both state-of-the-art and have been widely evaluated on a large number of revisions of real world projects [44]. Ekstazi, STARTS, and FLiRTS 2 use class-level RTS, i.e., they all identify changes at the class level and select every test class that traverses or depends on any changed class. Ekstazi uses dynamic analysis and STARTS uses static analysis of compiled Java code.

3.1 Experimental setup

We evaluated FLiRTS 2 on 13 subjects—those listed in Table 4 with a gray background—out of the 21 used to evaluate Ekstazi [27,44]. We explicitly excluded those subjects we used in the tuning process of FLiRTS 2 to avoid any possible bias in the experiment results. The revisions were selected using the same method described in Sect. 2.4.1 with the additional criterion that the subject should also run with STARTS on the chosen revisions. Moreover, we relaxed the constraint on the number of changes between a revision and its successor from greater than 3% of the classes to at least one class. The total number of selected revisions is 3331 for the 13 subjects. Table 6 reports the number of the selected revisions for each subject. We automatically generated the UML class diagrams from the 3331 revisions and identified the



adapted classes using the method described in Sect. 2.4.1. We ran Ekstazi and STARTS on the 3331 code-level revisions, and FLiRTS 2 on the corresponding models using the best configuration obtained during the tuning process.

The zip archives that contain the current implementation of FLiRTS 2, UML class diagrams generated from the code revisions, scripts to download the code revisions and run the experiment, and the experimental results are available at:

https://cazzola.di.unimi.it/flirts2.html

3.2 Results

Table 6 reports the RTS results in terms of safety violation, precision violation, and test suite reduction. Table 7 reports and compares the time spent in the RTS process by the 3 tools. Finally, Table 8 reports on the fault-detection ability experiment.

Safety violation. The median safety violation is zero (or close to zero) for all the subjects. Figure 7a shows that the average safety violation is below or close to 20% for all the subjects wrt. STARTS and for 12 out 13 subjects wrt. Ekstazi. Among all the 3331 revisions, the average safety violation of FLiRTS 2 was 18.88% wrt. Ekstazi and 16.53% wrt. STARTS. This indicates that FLiRTS 2 missed approximately the 18% of the test classes that were selected by the other tools on average. This is because FLiRTS 2 is designed to work with UML class diagrams and this diagram type cannot provide certain types of information used by the other tools such as (1) dynamic dependencies of test classes with the classes under test, (2) Java reflection, (3) exceptions, (4) dependencies from test classes to third party libraries and input configuration files, and (5) the code contained inside the method body (e.g., local variables referencing classes and method invocations on classes).

Precision violation. The median precision violation is zero for 10 out 13 subjects wrt. Ekstazi and 11 out 13 subjects wrt. STARTS, and never goes above 31% in the few remaining cases both wrt. Ekstazi and STARTS. Figure 7b shows that the average precision violation of FLiRTS 2 was below or close to 30% for mostly all the subjects wrt. both Ekstazi and STARTS. Among all the 3331 revisions, the average precision violation of FLiRTS 2 was 13.27% wrt. Ekstazi and 9.01% wrt. STARTS. This is because FLiRTS 2 is based on a probabilistic model, while Ekstazi is based on collecting the dynamic dependencies of the test classes, which can more precisely exclude test classes that do not traverse adapted classes. STARTS is a static approach and therefore less precise than Ekstazi. Thus, FLiRTS 2 achieved a lower average precision violation wrt. STARTS than wrt. Ekstazi.

Test suite reduction. Figure 7c plots the average percentage of selected test cases (100 minus the average reduction percentage from Table 6) by the 3 tools. It is evident

that (i) FLiRTS 2 achieved a reduction for all the subjects even though it is less precise and (ii) its reduction is comparable to the reduction achieved by the other tools. The average reduction percentage achieved by FLiRTS 2 over all the subjects was 82.06%. FLiRTS 2 achieved a higher or close (i.e., with a difference of less than 0.5%) average reduction percentage than STARTS on 10 out of 13 subjects. Even though Ekstazi is a dynamic approach, Ekstazi and FLiRTS 2 achieved a close (a difference of more or less 2%) average reduction percentage on 7 out of 13 subjects.

Time savings. During the running of the experiment, we also collected the time needed to select the test cases and for their execution. Table 7 reports such measurements. The data show that FLiRTS 2 tends to be faster than both Ekstazi and STARTS during the test case selection phase; over the 3331 considered revisions, on average FLiRTS 2 takes 11.83 s whereas Ekstazi takes 13.96 s and STARTS takes in average 102.58 s. On the other hand, considering also the time needed to execute the selected test cases the benefit is not that remarkable; over the 3331 considered revisions, on average FLiRTS 2 takes 253.80s whereas Ekstazi takes 238.53 s and STARTS takes 351.47 s. This is due to the fact that FLiRTS 2 is less precise than the other tools but despite this it is still faster than Ekstazi on 11 out of 13 of the considered subjects (the 12th column in Table 7) and it overperforms STARTS on all the subjects (the 13th column in Table 7). The 16th column in Table 7 makes explicit that FLiRTS 2 always saves a significant amount of time wrt. the execution of the whole test suite. The only exception is subject p_{21} where the limited test case reduction makes the time spent in selecting higher than the saved time; anyway FLiRTS 2 still behaves better than the other two tools because of its minimal selecting time.

Fault-detection ability. In the absence of real faults, mutation operators are often used to seed faults in testing experiments [5]. We use a mutation-based approach with the PIT [18] tool to apply first-order method-level mutation operators to the revisions of 12 subjects out of the 13 considered in the previous experiment (p₅ was removed because its JUnit tests are incompatible with PIT). We applied all the 13 PIT mutation operators [18] and configured PIT to mutate only the adapted classes of each revision. The adapted classes are detected by STARTS and used for the whole experiment. Moreover, the same mutations are used for all the three tools to avoid any bias introduced by different mutations.

For each revision, we ran PIT with the full test suites and those selected by FLiRTS 2, Ekstazi, and STARTS. Table 8 shows the experimental results. Note that PIT assigns 0 to the mutation score of a revision when the corresponding tool did not select any test case. This is because no test case can be used to kill the introduced mutants. We did not modify this behavior in the experiment.

The third column of Table 8 reports the average mutation score per subject achieved by running the full test suite, i.e.,



result
RTS
9
ble
ā

Diametro Action Actio																
bf 17 42 42 2.79 6.53 16.4 14.74 0 29.15 63.49 30.56 65.51 71.02 65.35 71.02<	9	SUA	Revs	Aad	Ate	A-SV _e (%)	A-SV _S (%)	M-SV _e (%)	M-SV _S (%)	A-PV _e (%)	A-PV _S (%)	M-PV _e (%)	M-PV _S (%)	A-TR _e (%)	A-TR _S (%)	A-TRf (%)
ac114d2 429 11.2 1.54 42.36 17.69 0.83 0.83 1.5 9.25 79.35<	p5	bf17e42	42	2.79	0.93	16.8	14.74	0	0	29.15	23.09	30.56	26.51	71.02	65.35	50.95
8645960 531 1.23 0.95 14.61 20.09 0 2.56 1.23 0.04 9.05 <t< td=""><td>9d</td><td>ac114d2</td><td>429</td><td>11.22</td><td>1.54</td><td>42.36</td><td>17.69</td><td>0.83</td><td>0</td><td>35.79</td><td>31</td><td>28.33</td><td>15</td><td>79.25</td><td>73.93</td><td>51.07</td></t<>	9d	ac114d2	429	11.22	1.54	42.36	17.69	0.83	0	35.79	31	28.33	15	79.25	73.93	51.07
559de2c5972.852.3710.412.5709.084.97009.24989.21b002dd421660713.0815.38006.077.523.4007.27965.32e4a663C3291.941.1421.522.99006.071.19.49009.339.23d344ebe3364.451.3515.7215.740010.165.9609.1190.59131e4922336.241.3517.410010.165.9609.369.3055e11045442.610.5517.0515.91018.18009.8.018.015868141381.20.5410.9110.91004.68009.8.187c89c9a31.20.5410.910004.68009.4.44	D7	86a5960	531	1.23	0.95	14.61	20.09	0	0	2.56	1.23	0	0	96.72	89.25	98.23
b0024d4 16 67 1.38 15.38 0 7.52 3.4 0 7.79 65.32 e4a6a3c 329 1.94 1.5 2.59 0 6.07 6.07 0.53 0 0 7.79 65.32 d34 ebe 3.5 1.54 1.57 15.74 0 0 10.16 5.95 0 0 9.33 0 9.33 0 9.12 9.33 9.23 9.23 9.24 9.36 9.36 9.36 9.36 9.36 9.36 9.36 9.36 9.36 9.36 9.36 9.36 9.36 9.36 9.36 9.36 9.36 9.37 9.37 9.37 9.37 9.37 9.37 9.37 9.37 9.37 9.38 9.37 9.38 9.37 9.38 9.37 9.38 9.37 9.37 9.37 9.37 9.37 9.37 9.37 9.37 9.37 9.37 9.37 9.37 9.37 9.37 9.37	8d	559de2c	297	2.85	2.37	10.4	12.57	0	0	80.6	4.97	0	0	92.49	89.21	91.14
e4a6a3c 329 1.94 1.14 21.5 22.99 0 6.07 6.07 6.53 0 9.33 9.33 9.33 9.32 d344ebe 336 445 1.35 15.72 15.74 0 0 11 9.49 0 0 9.12 9.59 131e492 23 2.16 1.5 18.74 0 0 10.16 5.96 0 9.12 9.59 65eff0 240 11.83 0.94 22.6 17.41 0 0 18.18 0 0 0 9.21 9.89 58c8141 38 2.24 0.84 10.91 10.91 0 4.68 0.43 0 9 9.17 89.97 7c89c9a 3 1 0 0 0 0 0 9 9.17 9.97	6d	b0024d4	216	6.07	1.05	13.08	15.38	0	0	7.52	3.4	0	0	72.79	65.32	74.72
d34ebe 336 4.45 1.35 15.74 0 0 11 9.49 0 0 91.12 90.59 131e492 23 2.16 1.58 18.76 9.36 0 10.16 5.96 0 0 9.12 9.69 65eff00 240 11.83 0.94 22.6 17.41 0 0 32.18 0 0 81.01 67.45 18.18 0 0 9.71 87.19 18.18 0 0 0 92.17 89.91 5868141 38 2.24 0.84 10.91 10.91 0 0 4.68 0.43 0 0 92.17 89.97 7c89c9a 3 1 0 0 0 0 0 0 0 92.17 89.97	D10	e4a6a3c	329	1.94	1.14	21.5	22.99	0	0	6.07	2.53	0	0	93.33	92.32	94.8
131e492 233 2.16 1.58 18.76 9.36 0 10.16 5.96 0 9.82.92 79.89 65eff00 240 11.83 0.94 22.6 17.41 0 0 18.18 0 0 81.01 67.45 b5101d5 44 2.61 0.55 17.05 15.91 0 0 18.18 0 0 0 18.18 18.18 0 0 18.18 <	p11	d344ebe	336	4.45	1.35	15.72	15.74	0	0	11	9.49	0	0	91.12	90.59	90.36
65effol 44 11.83 0.94 22.6 17.41 0 0 32.18 20.33 21.64 0 81.01 67.45 b5101d5 44 2.61 0.55 17.05 15.91 0 0 18.18 0 0 0 38.63 18.18 5868141 38 2.24 0.84 10.91 10.91 0 0 4.68 0.43 0 0 92.17 89.97 7c89c9a 3 1 0 0 0 0 0 0 44.44 22.22	D12	131e492	233	2.16	1.58	18.76	9:36	0	0	10.16	5.96	0	0	82.92	79.89	80.64
b5101d5 44 2.61 6.55 17.05 15.91 0 18.18 0 0 0 38.63 18.18 58.68141 38 2.24 0.84 10.91 10.91 0 0 31.29 25.57 0 0 92.17 89.97 243.4446 293 1.2 0.54 13.29 17.19 0 4.68 0.43 0 0 85.18 74.07 7c89c9a 3 1 0 0 0 0 0 0 44.44 22.22	P/4	65eff00	240	11.83	0.94	22.6	17.41	0	0	32.18	20.33	21.64	0	81.01	67.45	67.38
5868141 38 2.24 0.84 10.91 10.91 0 0 31.29 25.57 0 0 92.17 89.97 2434446 293 1.2 0.54 13.29 17.19 0 4.68 0.43 0 0 85.18 74.07 7c89c9a 3 1 0 0 0 0 22.22 0 0 44.44 22.22	D17	b5101d5	4	2.61	0.55	17.05	15.91	0	0	18.18	0	0	0	38.63	18.18	33.33
243446 293 1.2 0.54 13.29 17.19 0 0 4.68 0.43 0 0 85.18 74.07 7c89c9a 3 1 0 0 0 0 0 22.22 0 0 0 44.44 22.22	p18	5868141	38	2.24	0.84	10.91	10.91	0	0	31.29	25.57	0	0	92.17	89.97	83.32
7c89c9a 3 1 0 0 0 0 0 0 22.22 0 0 0 0 44.44 22.22	6/d	2434446	293	1.2	0.54	13.29	17.19	0	0	4.68	0.43	0	0	85.18	74.07	86.56
	D21	7c89c9a	3	1	0	0	0	0	0	22.22	0	0	0	44.44	22.22	22.22

SHA is the latest available revision used in the evaluation

Revs is the number of revisions for each subject

Aad (Atc) is the average number of adapted classes (test classes) among all classes (test classes) of all the revisions for each subject
 A- or M-SV₁ is the average/median (per subject) safety violation of FLiRTS2 with respect to tool i, i.e., Ekstazi and STARTS

A-TRe, A-TRs, and A-TRf are the average reductions (per subject) in test suite size achieved by Ekstazi, STARTS, and FLiRTS 2, respectively A- or M-PV_i is the average/median (per subject) precision violation of FLiRTS 2 with respect to tool i, i.e., Ekstazi and STARTS

Table 7 RTS performance (all data are in s)

<u>a</u>	A-rtse	A-sele	A-tote	A-rtse A-sele A-tote A-rtss	A-sel _s	A-tots	A-rtsf	A-self	A-totf	A-tot	gain f $ ightarrow$ e	gain f $ o$ s	gain e→ tot	gain s→ tot	gain f→ tot
p5	0.8845	4.7300	5.6145	6.7250	4.4018	11.1268	0.8950	4.0491	4.9441	6.3038	0.6704	6.1827	0.6893	-4.8230	1.3597
9d	0.9939	24.8960	25.8899	15.7795	22.923	38.7025	2.1413	19.6627	21.8040	34.8660	4.0859	16.8985	8.9761	-3.8365	13.0620
D7	0.9843	5.8391	6.8234	11.1404	7.4629	18.6033	1.8735	4.7945	0899.9	32.1116	0.1554	11.9353	25.2882	13.5083	25.4436
8d	0.9489	17.8280	18.7769	7.3797	21.7429	29.1226	1.1040	16.5697	17.6737	97.3553	1.1032	11.4489	78.5784	68.2327	79.6816
6d	0.9364	3.4569	4.3933	5.2169	3.3759	8.5928	0.5609	2.8167	3.3776	5.9529	1.0157	5.2152	1.5596	-2.6399	2.5753
D10	0.9841	5.0419	6.0260	7.0065	4.7841	11.7906	0.8514	4.2070	5.0584	9.5647	929670	6.7322	3.5387	-2.2259	4.5063
D11	1.0633		8.9471	7.6007	7.4772	15.0779	0.6918	6.3502	7.0420	12.7108	1.9051	8.0359	3.7637	-2.3671	5.6688
D12	1.0295	29.8194	30.8489	8.6726	29.2692	37.9418	0.8723	27.7132	28.5855	72.5338	2.2634	9.3563	41.6849	34.5920	43.9483
P14	0.9282		70.4401	5.2638	89.7286	94.9924	0.6799	91.0960	91.7759	171.1304	-21.3358	3.2165	100.6903	76.1380	79.3545
D17	1.3163	5.5765	6.8928	8.0330	5.1722	13.2052	0.5615	4.2990	4.8605	6.4985	2.0323	8.3447	-0.3943	-6.7067	1.6380
P18	0.9624		42.3643	10.2136	43.1718	53.3854	0.5933	51.4599	52.0532	158.3333	- 9.6889	1.3322	115.9690	104.9479	106.2801
P19	0.9453	3.3052	4.2505	5.2014	3.5176	8.7190	0.5196	2.5577	3.0773	8.6909	1.1732	5.6417	4.4404	-0.0281	5.6136
D21	1.9843	5.2757	7.2600	4.3457	5.8653	10.2110	0.4830	6.3987	6.8817	6.3730	0.3783	3.3293	-0.8870	-3.8380	-0.5087

A-sele, A-sels, and A-self is the average time (per subject) needed to run the test cases selected by Ekstazi, STARTS, and FLIRTS 2, respectively A-tote, A-tote, and A-totf is the average time (per subject) needed to select and run the test cases by Ekstazi, STARTS, and FLIRTS 2, respectively A-rtse, A-rtse, and A-rtse, and A-rtse is the average time (per subject) to select the test cases needed by Ekstazi, STARTS, and FLiRTS 2, respectively

A-tote, A-tote, and A-tote has the average time (per subject) needed to select and run the test cases by Exist

A-tot is the average time (per subject) needed to run all the test cases

gain e \rightarrow tot, gain s \rightarrow tot, and gain f \rightarrow tot is the average gain that Ekstazi, STARTS and FLIRTS 2, respectively, have over all test cases execution (a negative number represents a loss.) gain $\mathbf{f} + \mathbf{e}$ and gain $\mathbf{f} + \mathbf{s}$ is the average gain that FLiRTS 2 has over Ekstazi and STARTS, respectively (a negative number represents a loss.)



the average percentage of killed mutants wrt. the introduced mutants. The average mutation score for each subject is low because the available test suites for the considered subjects do not completely cover the subject code base. In particular the test suites lack the ability to cover the adapted portion of the code base where we introduced the mutations. It should be evident that no subset of the full test suite can kill more mutants than the whole test suite. Therefore, we consider the mutation score for the whole test suites as the term of comparison for the mutation scores achieved with the considered tools. The fourth, fifth, and sixth columns report the average deviation of the mutation score for FLiRTS 2, Ekstazi and STARTS, respectively. The deviation from the average for the subject s is calculated as

$$dev_t^s = (A-MS^s - A-MS_ts)^2$$

where **A-MS**^s is the average mutation score achieved by exercising the full test suite on all the revisions of the subject **s** and **A-MS**^s with **t** instantiated as FLiRTS 2, Ekstazi or STARTS is the average mutation score achieved by exercising the test suite selected by the tool **t** on the revisions of the subject **s**. Table 8 shows that all the selections have a low average deviation for every subject. In most cases—11, 10, and 6 out of 12 subjects for Ekstazi, STARTS, and FLiRTS 2, respectively—they deviate by approximately 1 percentage point and only in a few cases—1, 2, and 1 out of 12 for Ekstazi, STARTS, and FLiRTS 2, respectively—they deviate by more than 10 percentage points. The standard deviation of the averages for each tool **t**

$$\sigma_t = \sqrt{\frac{\sum_{s \in S} dev_t^s}{|S|}}$$

shows that all the tools deviate by less than 2 percentage points. In particular, the standard deviation for FLiRTS 2 is 1.78%, for Ekstazi is 1.11%, and for STARTS is 1.43%. Ekstazi has a higher chance to detect a fault than STARTS and FLiRTS 2 because it is a dynamic approach; FLiRTS 2 instead loses a few tenths of a percentage point because it selects fewer test cases than the other tools (as seen in the previous experiment) and the missed test cases include modification-traversing test classes that killed mutants in the adapted classes. In all three cases, the difference is negligible and should not compromise the fault detection ability of the tools.

The last three columns of Table 8 show how many cases (in percentage) FLiRTS 2 performs equal to or better than Ekstazi and STARTS in detecting a fault. For each subject, it shows the percentage of revisions in which the test classes selected by FLiRTS 2 killed at least as many as mutants than the test classes selected by Ekstazi and STARTS, respectively. The percentages in the 7th column show where the

test cases selected by FLiRTS 2 kill as many mutants as the whole test suite. On average, FLiRTS 2 always ranks above 94% in comparison with any test suite selection—94.88% wrt. the full test suite, 95.63% wrt. Ekstazi selection and 95.33% wrt. STARTS selection. For FLiRTS 2, the losses are mainly attributable to its safety violations wrt. Ekstazi and STARTS; it missed some modification-traversing test cases that could have detected the faults.

3.3 Threats to validity

The external and internal threats to validity to the tuning process also apply here. We mitigated them in the same way as before. Moreover, we used 3331 revisions from 13 open-source projects varying in size, application domain, and number of test classes.

Construct validity. In general, we could have used other metrics (e.g., test coverage) to evaluate the effectiveness of FLiRTS 2. However, we used the most common metrics in the research literature: safety violation, precision violation, reduction in test suite size, and fault-detection ability.

As per the time saving experiment, a clear problem is that the repeated execution of the same piece of code can provide different execution times due to several factors external to our control, including but not limited to memory paging, device and internet latencies, and CPU loading. As far as possible, we mitigated this issue by considering the computer load and by considering the average execution times rather than absolute values.

As per the fault detection ability experiment, one possible issue can be related to the fact that the randomly generated faults could be applied to code portions that are not adapted. We mitigated this issue by limiting the application of the mutation operators to those classes that STARTS detects as changed. The threat to construct validity is that we use mutation scores as a measure of fault detection ability. In the experiment, mutations may have been applied to code that was not modified in the revision. However, all the tools are equally affected because they all depend on identifying changes at the class level and they select any test case that traverses through the classes.

4 Related work

RTS has been studied for over three decades [24,24,76]. Below we summarize the existing code-based, model-based, and fuzzy logic-based approaches.

Code-based RTS. Kung et al. [41], Hsia et al. [35], and White and Abdullah [71] proposed firewall-based approaches. The firewall contains the changed classes and their dependent classes. Test cases that traverse classes in the firewall are selected. Skoglund and Runeson's [67] change-based



Table 8 Fault detection ability results

ID	Revs	A-MS (%)	dev _f (%)	dev _e (%)	$\operatorname{dev}_{\mathbf{s}}(\%)$	$f \geq full \ (\%)$	$\mathbf{f} \geq \mathbf{e} \ (\%)$	$f \ge s (\%)$
P6	429	3.02	10.6276	1.1881	10.6276	99.77	98.37	95.57
P 7	531	24.42	0.9801	0.09	0.0001	96.61	96.61	96.42
p ₈	597	23.41	3.8025	0.2704	0.4096	92.63	93.13	93.13
P9	216	18.05	8.8804	0	0	94.44	94.44	94.44
P10	329	18.93	0.0676	0	0.01	90.27	90.27	90.58
p ₁₁	336	34.65	3.2041	0.01	0.0001	96.73	97.02	97.32
P12	233	22.70	4.3681	1.1664	1.21	90.56	93.13	93.99
P14	240	9.49	0.0729	0.0036	0.1444	95.42	96.67	97.08
P17	44	19.34	4.7524	0.0004	0.0004	97.73	97.73	97.73
P18	38	40.47	0.0004	0.0004	0.0009	86.84	94.74	92.11
P19	293	9.60	1.1236	12.1801	12.1801	92.49	91.13	91.13
P21	3	57.33	0	0	0	100.00	100.00	100.00
		18.90	$\sigma_{\rm f} = 1.78$	$\sigma_{\rm e} = 1.11$	$\sigma_{\rm s}=1.43$	94.88	95.63	95.35

ID refers to Table 4, Revs is the number of revisions considered in the experiment

A-MS is the average mutation scores achieved by the full test classes, dev_f , dev_e , and dev_s are the deviations from A-MS of the mutation score of FLiRTS, Ekstazi, and STARTS, respectively

Columns $f \ge full$, $f \ge e$, and $f \ge s$ show the percentage of revisions in which the test classes selected by FLiRTS achieved a higher or equal mutation scores than all test classes and the test classes selected by Ekstazi and STARTS, respectively

Last row reports the average standard deviation for FLiRTS ($\sigma_{\mathbf{f}}$), Ekstazi ($\sigma_{\mathbf{e}}$) and STARTS ($\sigma_{\mathbf{s}}$) and the averages w.r.t. the total number of revisions for the last three columns

approach only selects those test cases that exercise the changed classes. ChEOPSJ [68,69] is a static change-based approach that uses the FAMIX model to represent software entities including test cases and building dependencies between them. These approaches use fine-grained information such as constructor calls and method invocation statements to build dependencies between software entities. FLiRTS 2 does not have, nor require, this information.

Rothermel and Harrold [61] analyze the program's control flow graph to identify changes at the statement level. Harrold et al. [33] extended the approach using the Java Inter-class Graph (JIG) to support Java. This is unlike FLiRTS 2, which identifies dependencies and changes at the class level.

Yu et al. [77] evaluated method-level and class-level static RTS in continuous integration environments. Class-level RTS was determined to be more practical and time-saving than method-level RTS. Gyori et al. [31] compared variants of dynamic and static class-level RTS with project-level RTS in the Maven Central open-source ecosystem. Class-level RTS was found to be an order of magnitude less costly than project-level RTS in terms of test selection reduction. Several other RTS approaches [27,45,81] were recently proposed to make RTS more cost-effective for modern software systems. FLiRTS 2 follows this recent trend [27,31,44,45,77] and focuses on class-level RTS.

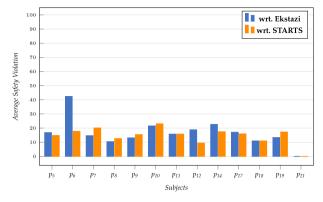
Ekstazi [27] is a change-based approach that tracks dynamic dependencies of test cases at the class level, and is formally proven to be safe. STARTS [44,45] is a static approach that builds a dependency graph of program types

based on compile-time information, and selects test cases that can reach changed types in the transitive closure of the dependency graph. HyRTS [81] is a dynamic and hybrid approach that supports analyzing the adapted classes at multiple levels of granularity (method and file) to improve the precision and selection time. Running HyRTS using the class-level mode produces the same RTS results as Ekstazi [81]. Thus, HyRTS was not consider in our evaluation.

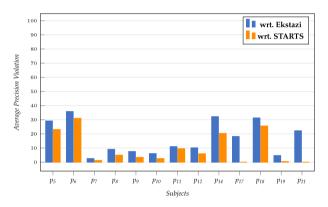
Other recent RTS approaches [6,59] are tunable to allow for trade-offs between safety and test selection reduction for situations where the regression testing budget is limited. SPIRITuS [59] uses information retrieval to select test cases, and can be easily adapted to different programming languages. ReTEST [6] is a language independent RTS approach that uses information retrieval to select test cases based on test failure history, test case diversity, and code change history at the line level. ReTEST eliminates the need for dynamic and static analysis. Unlike FLiRTS 2, SPIRITuS and ReTEST use fine-grained information from method bodies, test coverage, and failure history information. However, they are similar in that they are programming language independent and tunable with respect to different objectives.

Model-based RTS. Briand et al. [11] classify test cases based on changes performed to UML use case diagrams, class diagrams, and sequence diagrams. Farooq et al. [25] classify test cases based on changes made to UML class and state machine diagrams. Zech et al. [79,80] presented a generic model-based RTS platform controlled by OCL queries that can identify the changed elements of the models, which are

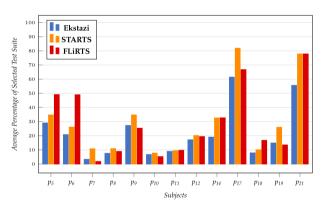




(a) Average safety violation



(b) Average precision violation



(c) Average percentage of selected test suite

Fig. 7 Averages of safety and precision violations, and test suite selection of FLiRTS 2

then used to select the test case models. Korel et al. [40], Ural et al. [70], and Lity et al. [48] proposed model-based RTS approaches based on state machine diagrams. Ye et al. [75] select test cases based on changes performed to UML activity diagrams. MaRTS [3] classifies test cases based on changes performed to UML class and executable activity diagrams. Unlike FLiRTS 2, the above model-based approaches use some type of behavioral model that must contain sufficient details to obtain the traceability links between the test

cases and the model elements, which is not common in practice [11].

Fuzzy logic-based RTS. Xu et al. [74] and Malz et al. [49] select test cases using code-based fuzzy logic approaches that require test coverage information.

Rapos et al. [57] proposed a model-based approach that uses fuzzy logic to prioritize test cases based on the symbolic execution tree for UML-RT state machine diagrams. Rhmann et al. [58] used fuzzy logic to prioritize test paths generated from a state machine diagram that represents the system under test. FLiRTS [2] refines the UML activity diagrams that represent behaviors of a software system at a high level of abstraction, and applies fuzzy logic to classify the test cases according to the probabilistic correctness values associated with the used refinements. Unlike FLiRTS 2, the model-based approaches use behavioral diagrams and obtain the test coverage at the model level.

5 Conclusions and future work

In this work, we investigated the use of fuzzy logic in regression test selection to bridge the abstraction gap between the source code and the UML class diagrams that model structural aspects of the system. Our approach, called FLiRTS 2, uses information about the classes that are adapted in a class diagram, and generates a class relationships graph where the nodes represent the classes and interfaces, and the weighted directed edges represent the likelihood that a class is calling an operation defined in another class. Paths from test cases to adapted classes are associated with varying degrees of certainty that they are actually exercised by some test case. FLiRTS 2 uses a probabilistic approach based on fuzzy logic to deal with the uncertainty and classify the test cases as retestable or reusable.

The key advantage of using FLiRTS 2 is that it relies only on class diagrams containing information that is commonly provided by software developers. It needs far less information than other model-based RTS approaches. Neither behavioral models, traceability links from the test cases to model elements, nor coverage data are needed. In spite of the limited information available to FLiRTS 2, it can still achieve test suite reduction without losing much fault detection effectiveness (a standard deviation from the whole test suite of 1.78 percentage points).

Using 13 subjects, we compared the safety violation, precision violation, test suite reduction, and fault detection ability of FLiRTS 2 with those of Ekstazi and STARTS. The average safety violations of FLiRTS 2 with respect to Ekstazi and STARTS were 18.88% and 16.53%, respectively. The average precision violations were 13.27% and 9.01%, respectively. The average test suite reduction using FLiRTS 2 was 82.06%.



We plan to explore search-based techniques to tune the fuzzy logic system. We also plan to explore the use of fuzzy logic (1) to select regression test cases based on changes in requirements models, and (2) to prioritize test cases based on objectives such as test case diversity. As machine learning techniques gain traction in software engineering research, it is worthwhile to investigate their use in RTS approaches.

Funding Open access funding provided by Università degli Studi di Milano within the CRUI-CARE Agreement. This research was supported in part by the US National Science Foundation (OAC-1931363 and IIS-2027550).

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

- Al-Alshuhai, A., Siewe, F.: An extension of class diagram to model the structure of context-aware systems. In: Proceedings AET'15, Cochin, India (2015)
- Al-Refai, M., Cazzola, W., Ghosh, S.: A fuzzy logic based approach for model-based regression test selection. In: MoDELS'17, pp. 55– 62. IEEE, Austin (2017)
- Al-Refai, M., Ghosh, S., Cazzola, W.: Model-based regression test selection for validating runtime adaptation of software systems. In: ICST'16, pp. 288–298. IEEE, Chicago (2016)
- Al-Refai, M., Ghosh, S., Cazzola, W.: Supporting inheritance hierarchy changes in model-based regression test selection. Softw. Syst. Model. 18(2), 937–958 (2019)
- Andrews, J.H., Briand, L.C., Labiche, Y.: Is mutation an appropriate tool for testing experiments? In: ICSE'05, pp. 402–411. IEEE, Saint Louis (2005)
- Azizi, M., Do, H.: ReTEST: a cost effective test case selection technique for modern software development. In: ISSRE'18, pp. 144–154. IEEE, Memphis (2018)
- Benselim, M.S., Seridi-Bouchelaghem, H.: Extended UML for the development of context-aware applications. In: Proceedings NDT'12, Communications in Computer and Information Science 293, pp. 33–43. Springer, Dubai (2012)
- 8. Bergmann, M.: An Introduction to Many-Valued and Fuzzy Logic: Semantics, Algebras, and Derivation Systems. Cambridge University Press, Cambridge (2008)
- Bock, C., Cook, S., Rivett, P., Rutt, T., Seidewitz, E., Selic, B., Tolbert, D.: About the unified modeling language specification Version 2.5.1. https://www.omg.org/spec/UML/2.5.1/ (2017)
- Boudjemline, H., Touahria, M., Boubetra, A., Kaabeche, H.: Heavyweight extension to the UML class diagram metamodel for modeling context aware systems in ubiquitous computing. J. Perv. Comput. Commun. 13(3), 238–251 (2017)

- Briand, L.C., Labiche, Y., He, S.: Automating regression test selection based on UML designs. J. Inf. Softw. Technol. 51(1), 16–30 (2009)
- Brun, C., Pierantonio, A.: Model differences in the eclipse modeling framework. UPGRADE Eur. J. Inform. Prof. 9(2), 29–34 (2008)
- Cazzola, W., Rossini, N.A., Al-Refai, M., France, R.B.: Fine-grained software evolution using UML activity and class models.
 In: MoDELS'13, LNCS 8107, pp. 271–286. Springer, Miami (2013)
- Cazzola, W., Rossini, N.A., Bennett, P., Pradeep Mandalaparty, S., France, R.B.: Fine-grained semi-automated runtime evolution. In: MoDELS@Run-Time, LNCS 8378, pp. 237–258. Springer (2014)
- Chen, Y., Probert, R.L., Sims, D.P.: Specification-based regression test selection with risk analysis. In: CASCON'02, pp. 1–14. IBM Press (2002)
- Cingolani, P., Alcalaá-Fdez, J.: jFuzzyLogic: a java library to design fuzzy logic controllers according to the standard for fuzzy control programming. Int. J. Comput. Intell. Syst. 6(1), 61–75 (2013)
- 17. Cintra, M.E., Camargo, H.A., Monard, M.C.: A study on techniques for the automatic generation of membership functions for pattern recognition. In: Proceedings of the 3rd Conference of Trinational Academy of Sciences, pp. 1–10. Foz de Iguaçu, Brazil (2008)
- Coles, H., Laurent, T., Henard, C., Papadakis, M., Ventresque, A.: PIT: a practical mutation testing tool for java. In: ISSTA'16, pp. 449–452. ACM, Saarbrücken, Germany (2016)
- Cordón, O., Gomide, F., Herrera, F., Hoffmann, F., Magdalena, L.: Ten years of genetic fuzzy systems: current framework and new trends. Fuzzy Sets Syst. 141(1), 5–31 (2004)
- Deza, M.M., Deza, E.: Encyclopedia of Distances. Springer, Berlin (2009)
- Dobing, B., Parsons, J.: current practices in the use of UML. In: ER'05, pp. 2–11. Klagenfurt, Austria (2005)
- Dobing, B., Parsons, J.: How UML is used. Commun. ACM 49(5), 109–113 (2006)
- Dzidek, W.J., Arisholm, E., Briand, L.C.: A realistic empirical evaluation of the costs and benefits of UML in software maintenance. IEEE Trans. Softw. Eng. 34(3), 407–432 (2008)
- Engström, E., Runeson, P., Skoglund, M.: A systematic review on regression test selection techniques. Inf. Softw. Technol. 52(1), 14– 30 (2010)
- Farooq, Q.U.A., Iqbal, M.Z.Z., I Malik, Z., Riebisch, M.: A model-based regression testing approach for evolving software systems with flexible tool support. In: Proc. ECBS'10, pp. 41–49. IEEE, Oxford (2010)
- Fernández-Sáez, A.M., Caivano, D., Genero, M., Chaudron, M.R.V.: On the use of uml documentation in software maintenance: results from a survey in industry. In: MODELS'15, pp. 292–301. IEEE, Ottawa, ON (2015)
- Gligoric, M., Eloussi, L., Marinov, D.: Practical regression test selection with dynamic file dependencies. In: ISSTA'15, pp. 211– 222. ACM, Baltimore, MD (2015)
- Goyal, V.: Analyze your classes. O'Reilly (2003). http://www. onjava.com/pub/a/onjava/2003/10/22/bcel.html?page=1
- Gregersen, A.R., Jørgensen, B.N.: Dynamic update of java applications-balancing change flexibility vs programming transparency. J. Softw. Maint. Evol. Res. Pract. 21(2), 81–112 (2009)
- Grossman, M., Aronson, J.E., McCarthy, R.V.: Does UML make the grade? Insights from the software development community. Inf. Softw. Technol. 47(6), 383–397 (2005)
- Gyori, A., Legunsen, O., Hariri, F., Marinov, D.: Evaluating regression test selection opportunities in a very large open-source ecosystem. In: ISSRE'18, pp. 112–122. IEEE, Memphis, TN (2018)



- 32. Harrold, M.J.: Testing evolving software. J. Syst. Softw. **47**(2–3), 173–181 (1999)
- Harrold, M.J., Jones, J.A., Li, T., Liang, D., Orso, A., Pennings, M., Sinha, S., Spoon, S.A., Gujarathi, A.: Regression test selection for java software. In: OOPSLA'01, pp. 312–326. ACM, Tampa, FL (2001)
- Hdaytullah, Vathsavayi, S., Räihä, O., Koskimies, K., Gregersen, A.R.: Applying genetic self-architecting for distributed systems. In: Proceedings of the 4th World Congress on Nature and Biologically Inspired Computing (NaBIC'12), pp. 44–52. IEEE, Mexico City (2012)
- Hsia, P., Li, X., Kung, D.C.H., Hsu, C.T., Li, L., Toyoshima, Y., Chen, C.: A technique for the selective revalidation of OO software. J. Softw. Evol. Process 9(4), 217–233 (1997)
- Hsu, I.C.: Extending UML to model web 2.0-based context-aware applications. Softw. Pract. Exp. 42(10), 1211–1227 (2012)
- Hutchinson, J., Whittle, J., Rouncefield, M., Kristoffersen, S.: Empirical assessment of MDE in industry. In: ICSE'11, pp. 471–480. IEEE, Honolulu, HI (2011)
- Jensen, A.C., Cheng, B.H.C.: On the use of genetic programming for automated refactoring and the introduction of design patterns. In: GECCO'10, pp. 1341–1348. ACM, Portland, OR (2010)
- Kapitsaki, G.M., Venieris, I.S.: PCP: privacy-aware context profile towards context-aware application development. In: Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services (iiWAS'08), pp. 104–110. ACM, Linz (2008)
- Korel, B., Tahat, L.H., Vaysburg, B.: Model based regression test reduction using dependence analysis. In: ICSM'02, pp. 214–223. IEEE, Montréal, Quebec (2002)
- Kung, D.C., Gao, J., Hsia, P., Toyoshima, Y., Chen, C.: On regression testing of object-oriented programs. J. Syst. Softw. 32(1), 21–40 (1996)
- Lange, C.F.J., Chaudron, M.R.V., Muskens, J.: In practice: UML software architecture and design description. IEEE Softw. 23(2), 40–46 (2006)
- 43. Langer, P., Mayerhofer, T., Wimmer, M., Kappel, G.: On the usage of UML: initial results of analyzing open UML models. In: Proceedings of Modellierung (Modellierung'14), Lecture Notes in Informatics 225, pp. 289–304. Springer, Wien (2014)
- Legunsen, O., Hariri, F., Shi, A., Lu, Y., Zhang, L., Marinov, D.: An extensive study of static regression test selection in modern software evolution. In: FSE'16, pp. 583–594. ACM, Seattle, WA (2016)
- Legunsen, O., Shi, A., Marinov, D.: STARTS: static regression test selection. In: ASE'17, pp. 949–954. IEEE Press, Urbana-Champaign, IL (2017)
- Leroux, D., Nally, M., Hussey, K.: Rational software architect: a tool for domain-specific modeling. IBM Syst. J. 45(3), 555–568 (2006)
- 47. Leung, H.K.N., White, L.J.: A cost model to compare regression test strategies. In: ICSM'91, pp. 201–208. IEEE, Sorrento (1989)
- Lity, S., Morbach, T., Thüm, T., Schaefer, I.: Applying incremental model slicing to product-line regression testing. In: ICSR'16, LNCS 9679, pp. 3–19. Springer, Limassol (2016)
- Malz, C., Jazdi, N., Göhner, P.: Prioritization of test cases using software agents and fuzzy logic. In: ICST'12, pp. 483–486. IEEE, Montréal, BC (2012)
- 50. Mamdani, E.H., Assilian, S.: An experiment in linguistic synthesis with a fuzzy logic controller. Int. J. Man Mach. Stud. **7**(1), 1–13 (1975)
- Maoz, S., Ringert, J.O., Rumpe, B.: ADDiff: semantic differencing for activity diagrams. In: ESEC/FSE'11, pp. 179–189. ACM, Szeged (2011)
- Moghadam, I.H., Ó Cinneéide, M.: Automated refactoring using design differencing. In: CSMR'12, pp. 43–52. IEEE, Szeged (2012)

- 53. Naveh, B., Sichi, J.: JGraphT a free java graph library (2011)
- Osman, M.H., Chaudron, M.R.V.: UML usage in open source software development: a field study. In: Proceedings of 3rd International Workshop on Experiences and Empirical Studies in Software Modeling (EESSMod'13), pp. 23–32. Miami (2013)
- 55. Petre, M.: UML in practice. In: ICSE'13, pp. 722–731. IEEE, San Francisco (2013)
- Pinto, R.P., Cardozo, E., Coelho, P.R.S.L., Guimarães, E.G.: A domain-independent middleware framework for context-aware applications. In: Proceedings Workshop on Adaptive and Reflective Middleware (ARM'07), pp. 5:1–5:6. ACM, Newport Beach (2007)
- Rapos, E.J., Dingel, J.: Using fuzzy logic and symbolic execution to prioritize UML-RT test cases. In: ICST'15. IEEE, Graz (2015)
- Rhmann, W., Saxena, V.: Fuzzy expert system based test cases prioritization from UML state machine diagram using risk information. Int. J. Math. Sci. Comput. 3(1), 17–27 (2017)
- Romano, S., Scanniello, G., Antoniol, G., Marchetto, A.: SPIR-ITuS: a simple information retrieval regression test selection approach. Inf. Softw. Technol. 99, 62–80 (2018)
- Ross, T.J.: Fuzzy Logic with Engineering Applications, 3rd edn. Wiley, Hoboken (2011)
- Rothermel, G., Harrold, M.J.: A safe, efficient regression test selection technique. ACM Trans. Softw. Eng. Methodol. 6(2), 173–210 (1997)
- Sheng, Q.Z., Benatallah, B.: ContextUML: A UML-based modeling language for model-driven development of context-aware web services. In: Proceedings of the International Conference on Mobile Business (ICMB'05), pp. 206–212. IEEE, Sydney (2005)
- Shi, A., Gyori, A., Gligoric, M., Zaytsev, A., Marinov, D.: Balancing trade-offs in test-suite reduction. In: FSE'14, pp. 246–256. ACM, Hong Kong (2014)
- 64. da Silva, J.P.S., Ecar, M., Pimenta, M.S., Guedes, G.T.A., Franz, L.P., Marchezan, L.: A systematic literature review of UML-based domain-specific modeling languages for self-adaptive systems. In: SEAMS'18, pp. 87–93. ACM, Gothenburg (2018)
- 65. da Silva, J.P.S., Ecar, M., Pimenta, M.S., Guedes, G.T.A., Rodrigues, E.M.: Towards a domain-specific modeling language for self-adaptive systems conceptual modeling. In: U. Kulesza (ed.) Proceedings of the XXXII Brazilian Symposium on Software Engineering (SBES'18), pp. 208–213. ACM, Sao Carlos (2018)
- Simons, C., Wirtz, G.: Modeling context in mobile distributed systems with the UML. J. Vis. Lang. Comput. 18, 420–439 (2007)
- Skoglund, M., Runeson, P.: Improving class firewall regression test selection by removing the class firewall. Int. J. Softw. Eng. Knowl. Eng. 17(3), 359–378 (2007)
- Soetens, Q.D., Demeyer, S., Zaidman, A.: Change-based test selection in the presence of developer tests. In: CSMR'13, pp. 101–110. IEEE, Genoa (2013)
- Soetens, Q.D., Demeyer, S., Zaidman, A., Pérez, J.: Change-based test selection: an empirical evaluation. Empir. Softw. Eng. 21, 1990–2032 (2015)
- Ural, H., Yenigün, H.: Regression test suite selection using dependence analysis. J. Softw. Evol. Process 25, 681–709 (2013)
- White, L.J., Abdullah, K.: A firewall approach for regression testing of object-oriented software. In: Proceedings of the 10th International Software Quality Week (QW'97). San Francisco, CA (1997)
- Wiedermann Agner, L.T., Wisniewski Soares, I., Stadzisz, P.C., Simão, J.M.: A Brazilian survey on UML and model-driven practices for embedded software development. J. Syst. Softw. 86(4), 997–1005 (2013)
- Xing, Z., Stroulia, E.: Differencing logical UML models. Autom. Softw. Eng. 14(2), 215–259 (2007)
- Xu, Z., Gao, K., Khoshgoftaar, T.M., Seliya, N.: System regression test planning with a fuzzy expert system. Inf. Sci. 259, 532–543 (2014)



75. Ye, N., Chen, X., Jiang, P., Ding, W., Li, X.: Automatic regression test selection based on activity diagrams. In: Proceedings of the 5th International Conference on Secure Software Integration & Reliability Improvement Companion (SSIRI-C'11), pp. 166–171. IEEE, Jeju Island (2011)

- Yoo, S., Harman, M.: Regression testing minimization, selection and prioritization: a survey. J. Softw. Test. Verif. Reliab. 22(2), 67–120 (2012)
- Yu, T., Wang, T.: A study of regression test selection in continuous integration environments. In: ISSRE'18, pp. 135–143. IEEE, Memphis, TN (2018)
- 78. Zadeh, L.A.: Fuzzy sets. Inf. Control **8**(3), 338–353 (1965)
- Zech, P., Felderer, M., Kalb, P., Breu, R.: A generic platform for model-based regression testing. In: ISoLA'12, LNCS 7609, pp. 112–126. Springer, Heraclion (2012)
- Zech, P., Kalb, P., Felderer, M., Atkinson, C., Breu, R.: Model-based regression testing by OCL. Int. J. Softw. Tools Technol. Transf. 19(1), 115–131 (2017)
- Zhang, L.: Hybrid regression test selection. In: ICSE'18, pp. 199– 209. IEEE, Gotheburg (2018)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Walter Cazzola is an Associate Professor in the Computer Science Department of the Università degli Studi di Milano, Italy and the Chair of the ADAPT laboratory. Dr. Cazzola designed the mChaRM framework, @Java, [a]C#, Blueprint programming languages and he is involved in the designing and development of the Neverlang language workbench. He also designed the JavAdaptor dynamic software updating framework and its front-end FiGA. He has written over 100 scientific

papers. His research interests include (but are not limited to) software maintenance, evolution and comprehension, programming methodologies and languages. He served on the program committees or editorial boards of the most important conferences and journals about his research topics. He is associate editor for the Journal of Computer Languages published by Elsevier. More information about Dr. Cazzola and all his publications are available at https://cazzola.di.unimi.it and he can be contacted at cazzola@di.unimi.it for any question.



Sudipto Ghosh is a Professor of Computer Science at Colorado State University, USA. He received the Ph.D. degree in Computer Science from Purdue University, USA, in 2000. His teaching and research interests include model-based software development and software testing. He is on the editorial boards of IEEE Transactions on Reliability, Information and Software Technology, and Software Quality Journal. More information about Dr. Ghosh and all his publications are available from his

web page at http://www.cs.colostate.edu/~ghosh, and he can be contacted at ghosh@colostate.edu for any question.



Mohammed Al-Refai is an Assistant Professor of Computer Science at the University of Science and Technology in Jordan. He received the Ph.D. degree in Computer Science from Colorado State University in 2019. His research interests are model-based validation and regression test selection. He received his Bachelor's and Master's degrees in Computer Science from Jordan University of Science and Technology, Jordan. He can be contacted at mnalrefai@just.edu.jo.



Gabriele Maurina is a Master's student in Computer Science at the Università degli Studi di Milano, Italy. He received his Bachelor's degree in Computer Science from the Università degli Studi di Trento, Italy. He won a 6-month scholarship from the Università degli Studi di Milano to work on his Master's thesis at Colorado State University. His research interests are software testing and machine learning techniques. He can be contacted at gabriele. maurina@studenti.unimi.it.

