DOI: 10.15439/2023F5070
ISSN 2300-5963 ACSIS, Vol. 35

Investigating the Effect of Partial and Real-Time Feedback in INMAP Code-To-Architecture Mapping

Zipani Tom Sinkala, Sebastian Herold
0000-0002-7288-5552
0000-0002-3180-9182
Department of Mathematics and Computer Science, Karlstad University, Karlstad, Sweden
Email: {tom.sinkala, sebastian.herold}@kau.se

***Abstract*—InMap is an interactive and iterative information retrieval-based automated mapping algorithm that produces *code-to-architecture* mapping recommendations. In its original form, InMap requires an architect to provide feedback for each *code-to-architecture* mapping recommendation in a given set produced (*complete feedback*). However, architects may delay/ defer deciding on some of the mapping recommendations provided. This leads us to ask, how would InMap perform if only a subset of the recommendations provided *(partial feedback)* or only a single recommendation *(real-time feedback)* is reviewed by the architect? Through carefully designed mapping experiments, we show that an architect giving *partial* or *real-time feedback* does not harm the recall and precision of the recommendations produced by InMap. On the contrary, we observed from the results of the systems tested a net increase of 2-5% (depending on the approach). This shows that in addition to InMap's original *complete feedback* approach, the two new approaches of collecting feedback presented in this paper, i.e. *partial* and *real-time,* create flexibility in how software architecture consistency checking tool developers may choose to collect mapping feedback and how architects may opt-to provide feedback, with no harm to the recall and precision of the results.**

***Index Terms*—software architecture conformance, automated source code mapping, software architecture consistency, software maintenance.**

I. Introduction

SOFTWARE engineering industry practitioners use *Software Architecture Consistency Checking (SACC)* to check the consistency of a software's implementation with its architectural design [1, 8, 9, 17]. Its importance is derived from the fact that if a software system's implementation no longer conforms to its originally designed architecture, this may lead to failure in fulling its architectural design goals or in meeting intended software quality attributes such as reliability, availability, performance or security [1, 7, 15, 17].

Reflexion Modelling is an effective SACC technique as far as industry acceptance and tool support are concerned [6, 10, 12, 14]. It represents software architecture as a decomposition of a software system into *sub-components/architectural modules* and the *relationships/dependencies* allowed among them [10]. It maps the existing codebase onto the defined architectural modules revealing any irregular dependencies amongst the modules. If no irregular dependencies exist, the system's codebase is said to conform to its architecture. However, if dependencies not prescribed in the architecture exist, then the software's codebase is said to have diverged from its intended architecture – a phenomenon known as *architecture drift* or *architecture degradation* [10, 14, 16].

Mapping code to architecture in Reflexion Modelling is a tedious manual process, especially for large systems [1]. Nevertheless, techniques exist that attempt to 'automate' *code-to-architecture* mapping in SACC methods [3, 5, 12, 18]. Accomplishing this well is no trivial task, as ideally, mapping is done by a system expert knowledgeable about the system, its architecture and its codebase [10]. However, these techniques attempt to correctly predict which architectural module a *portion of the code* would be mapped to. They use various approaches to try and tackle this problem – from code dependency and clustering [5, 11] to machine learning, information retrieval and natural language processing techniques [3, 12, 19, 21, 22].

Furthermore, mapping techniques also differ in how an architect is involved in *verifying* the correctness of the resulting mapping. Some techniques only involve the architect after a complete mapping of the codebase, meaning the architect is given a complete mapping and then can decide if it is correct [3, 11, 12]. Other mapping techniques do it progressively or *interactively* in a *human-in-the-loop* approach. This means that as the mapping takes place, the architect is asked to review the mapping suggestions provided to improve them [19, 20] or resolve the cases that are difficult to map automatically [5].

Interactive mapping techniques have a similar approach to obtaining feedback from an architect as mapping progresses. They require an architect to decide on the correctness of every member of a set of mapping recommendations given. However, it is common to have cases where an architect may need more knowledge of a system [3, 5, 19, 21, 22] and may therefore prefer to *delay/defer* deciding on some mapping recommendations provided. In other words, given a set of 10 mapping recommendations, it is expected to have the architect decide about all 10 before progressing with the mapping. However, the limitation of this approach is that it does not accommodate a situation where an architect may feel that they are only able to make a decision about some of the recommendations produced (not all) or a case where the architect would want to defer deciding on a given recommendation to a later point in the mapping exercise. We thus investigate the effect, in terms of the impact on the recall and precision of the results, of providing *partial feedback* in a code-to-architecture mapping technique we developed in prior studies called InMap [18, 19, 20, 21, 22]. Additionally, we are interested in investigating InMap's behaviour if we update the list of recommendations provided in *real-time*; that is, the cases where a new set of mapping recommendations are provided each time an architect gives feedback for a singular recommendation.

Our contribution through this paper is evidence that an architect giving *partial* or *incomplete feedback* for the mapping recommendations produced by InMap does not harm the recall and precision of the technique's recommendations. InMap, in this modified form, retained recall values similar to its original *complete-feedback* form for all six systems under test. Additionally, we found that the precision of the *partial-feedback* approach is, on average, +/- 2% compared to the *complete-feedback* approach. We also show that updating InMap's mapping recommendations in *real-time*, as an architect provides feedback, gives a net increase of 5% for the systems tested. This entails that in addition to InMap's original complete feedback approach, the two new approaches of collecting feedback introduced in this paper, i.e. partial batch and real-time, provide flexibility in how SACC tool developers may choose to collect mapping feedback from an architect when using *human-in-the-loop* mapping techniques. We also show that our InMap mapping technique offers flexibility in how architects may opt to provide feedback with no harm to the recall and precision of the results.

Section II highlights related works. Section III gives an overview of InMap. In Sec IV, we discuss our new approaches to collecting feedback in-depth. Section V explains our experimental setup. The results of the experiments are presented in Section VI and discussed in Section VII. The paper is concluded in Section VIII.

II. Related Work

A few techniques exist that attempt to (auto)-map source code to software architecture in SACC methods like Reflexion Modelling. We broadly classify these into two categories with regard to how they collect feedback from an architect. We classify, as *collaborative mapping* or *just-in-time feedback*, techniques involving the architect as the mapping occurs. These techniques are incremental and involve a *human-in-the-loop.* They get feedback from the architect as the mapping progresses on the premise that the architect's knowledge of the software system can help 'steer' the mapping in the right direction [19, 20]. We classify, as *non-collaborative mapping* or *feedback after-the-fact*, those techniques that attempt to entirely automate the mapping process without involving the architect during the mapping process. An architect only reviews the final results of these techniques after they complete their mapping process, implying the architect is not directly involved in the mapping [7].

A. Collaborative Mapping / Just-In-Time Feedback

Christl et al. propose **HuGME,** a mapping recommendation technique that analyses source code elements' dependencies. HuGME clusters source code elements using an architect's knowledge about its intended architecture [4, 5]. A *dependency-based attraction function*, which minimises coupling and maximises cohesion, is used, which yields a matrix of attraction scores for unmapped entities [23]. All unmapped entities that result in only one candidate having a score higher than the mean of all scores result in a sole recommendation. All unmapped entities with two or more mapping candidates are presented to the architect in descending order as recommendations. HuGME presents the recommendations to the architect to let cluster decisions be made entirely by the architect. HuGME does not attempt to map all source code entities in one complete step; instead, it maps a subset at a time, getting feedback from the architect until no more mapping is possible. This classifies it as a *collaborative mapping technique*. HuGME needs about 20% of a system's codebase to be manually *pre-mapped* by an architect before proceeding with automated mapping and thus suffers from pre-mapping drawbacks [18, 19].

Bittencourt et al. have a mapping recommendation technique that uses *information retrieval (IR).* It has a similar automated mapping approach to HuGME, except that they use *dependency-based attraction functions* with an *IR-based similarity function* [3]. They compute the similarity of an unmapped entity to an architectural module by searching for specified terms within the source of an unmapped entity. They search for an architectural module's name and the names of its mapped entities/classes, class methods and fields. Their technique requires manual pre-mapping before it can automate mapping; hence it suffers from pre-mapping drawbacks similar to HuGME [18, 19]. The results of Bittencourt et al.'s technique show that when there was a smaller pre-mapped code base, there was a decrease in the f_1*-score* of their technique [3].

Naim et al. propose a technique that uses *dependency analysis* and *information retrieval* methods, called ***Coordinated Clustering of Heterogeneous Datasets (CCHD),*** to compute a *similarity score* for source entities [11]. CCHD profits from an architect's feedback on a recovered architecture to iteratively adjust the results until there are no more recommendations for change. These modified results train a classifier that automatically places new code in the "right" architectural module. However, the technique is not necessarily meant for automated mapping in *software architecture consistency checking,* but rather, it was designed for *software architecture recovery* tasks.

None of the above-discussed collaborative mapping approaches directly addresses the different ways an architect can provide feedback as the mapping progresses, namely *complete, partial or real-time.* They all make use of a complete-batch feedback approach for the recommendations they provide.

B. Non-Collaborative Mapping / Feedback After-the-Fact

Olsson et al. use *information retrieval* and *dependency analysis* in their automated mapping technique called **Naive Bayes Classification (NBC)** [12]. They use Bayes' theorem to build a probabilistic model of classifications using words taken from the source entities of a software system. The model provides the probability of words (or tokens) being part of a source entity. This is then enriched with syntactical information on a source entity's incoming and outgoing dependencies, a method called *Concrete Dependency Abstraction* [12]. NBC needs a pre-mapped set to return fruitful results and inadvertently suffers from the downsides that come with the need to pre-map. In recent studies, Olsson et al. refined their technique to create a pre-mapped set using an approach similar to InMap [13]. However, the technique uses a *feedback after-the-fact* approach, implying the architect checks whether the mapping is correct at the end of the process.

C. Manual Methods Supported by Tools

Naming patterns (or *regular expressions*) are commonly used in SACC industry tools. Expressions such as ***/cli/*** or **.cli.* or net.commons.cli.** can be used to map source entities (whether classes or packages) to an architecture module named *CLI.* This approach is used in popular SACC tools such as **Sonargraph Architect** and **Structure101 Studio.** These tools also provide *drag & drop* functionality. However, the limitation of using naming patterns or drag & drop functionality is that they do not solve the problem of reducing the monotony of the mapping process because they are both manual tasks. In large systems with complex mapping configurations, this is a demanding task.

In summary, there exist techniques that attempt to get feedback as the mapping progresses, like HuGME [4, 5] and InMap [19, 21, 22] and others that get feedback from the architect once the mapping task is complete, like NBC [12, 13]. Of those that collect feedback as the mapping progresses, none directly address the alternative ways architects may provide feedback, for example, as a *complete batch,* as a *partial batch* or *as a single recommendation* at a time.

III. InMap Interactive Mapping

In [18, 19], we propose a collaborative mapping technique known as InMap. Using natural language descriptions of a system's architecture modules, InMap can automate mapping a completely unmapped system with no loss in the recall and precision of the recommendations produced [19]. It iteratively and interactively provides mapping recommendations that an architect can review, a batch/set at a time. It uses the architect's feedback in one iteration to guide the recommendations provided in a subsequent iteration. This process continues until the entire system is mapped or no more recommendations can be produced. In our prior studies, InMap automated the mapping of completely unmapped systems with an average recall and precision of 0.97 and 0.82, respectively, for the systems tested. However, InMap does not cater for the fact that an architect may not have full knowledge of the set of recommendations provided and may instead opt to provide partial feedback on the recommendations given. Instead, it assumes or requires that the architect provides feedback for all recommendations produced in a given iteration/batch before a new set of recommendations can be provided in the following iteration.

A. The Algorithm

The InMap mapping technique is comprised of seven steps [19, 21] summarised as follows:

1. A software system's source files are filtered to omit third-party package libraries or system packages/classes that the architect does not want to be part of the mapping exercise.

2. The contents of the filtered source files are stripped of any special characters and programming language-specific keywords.

3. The pre-processed source files are indexed as an inverted index.

4. InMap constructs a query for each architectural module.

5. InMap uses the queries from *Step 4* to search the indexed source files for the similarity of every unmapped class to each architectural module. The query for each architectural module is a combination of (i) its name; (ii) its natural language architectural description; (iii) the names of classes mapped to the module; and (iv) the names of methods contained within classes mapped to the module. In InMap's first iteration, when there are no mapped classes, it uses only information from items (i) and (ii) to construct the query. However, once the first set of classes is mapped, InMap adds items (iii) and (iv) to the query. These last two items 'enrich' the query, as it were, to search for the similarity of any unmapped class to the architectural module in question. Consequently, after each iteration of newly mapped classes, the query to produce the next set of recommendations is different. The queries are used to search the index, resulting in a set of scores for every *class-module pair.* The scores are based on the similarity information retrieval function, *tf-idf.* The *tf-idf* scores

are called *class-to-module similarity scores (SS_{cm})*, where *c* and *m* are a class-module pair in the system under review. *Step 5* results in a matrix of *class-to-module similarity scores (SS_{cm})* for every class against every module. We extended InMap to include hierarchical information contained in a system's codebase, i.e., packages [20, 21], to condense the number of recommendations made to complete the mapping process in SACC techniques. This version of the technique in step five produces a matrix of *package-to-module similarity scores SS_{pm}* derived from *class-to-module similarity scores (SS_{cm})*.

6. InMap uses the matrix of *entity-to-module* scores to produce an ordered list of the best-scoring entities for the given architectural modules in terms of similarity. In this case, an entity is either a class or a package. InMap also uses page size to trim the ordered list to the most likely correct recommendations.

7. The architect reviews the recommendations produced, giving feedback by accepting or rejecting them. After this step, InMap returns to *Step 4* and iterates *Step 4* through *7* until no more recommendations can be produced.

At the time of our study, InMap existed in two versions, a class-based version and a package-based (or hierarchical) version. More detailed descriptions of both mapping algorithms and how their similarity score calculations are derived can be found in [19] and [21], respectively.

B. Research Questions

For our study, we hypothesise that there is more than one way a software architect may choose to provide feedback about code-to-architecture mapping recommendations produced by interactive techniques like InMap. They may do it as a *complete batch* which is how InMap works in its original form [19, 21, 22], but they may also do it as a *partial batch.* For example, if presented with a page of recommendations, the architect might be unsure about a few of them and would opt to postpone making a decision about them. This leads to asking,

> ***RQ1****: What is the effect, in terms of recall and precision, of an architect giving partial batch-feedback in InMap compared to complete batch-feedback?*

Another interesting scenario to investigate is the implication of InMap collecting and updating its list of recommendations in real-time. Rather than waiting for an architect to give feedback for all recommendations provided on a page before presenting a new set of recommendations (what we would describe as an *interactive batch update process*), what would be the behaviour of InMap, in terms of recall and precision if it updated its list of recommendations immediately an architect gives feedback on an individual recommendation (what we would describe as an *interactive real-time update process*)? In other words,

> ***RQ2****: What is the effect, in InMap in terms of recall and precision, to consider feedback from the architect as soon as we receive it (real-time feedback), and how does it compare with batch feedback?*

IV. Method

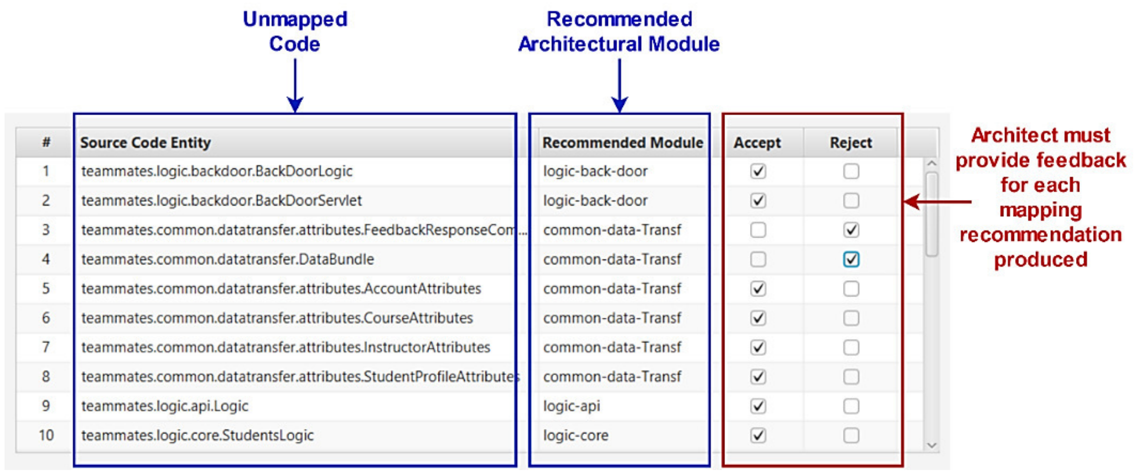
To properly investigate our research questions, we describe and formally define all three highlighted approaches to collecting feedback from an architect, the prior existing *complete-batch feedback* approach, as well as the two new approaches introduced in this study, namely *partial-batch feedback* and *real-time feedback*. We also illustrate how InMap was modified to accommodate the two new feedback approaches.

A. Complete-Batch Feedback

Complete-batch feedback is used by most interactive mapping techniques [3, 5, 19, 20]. The algorithm gets complete feedback from the architect for all mapping recommendations on a page – see *Fig. 1* for an illustration. The mapping algorithm only generates a new set/page of mapping recommendations in a subsequent iteration once feedback is given for all entities in the given set of the current iteration. This implies:

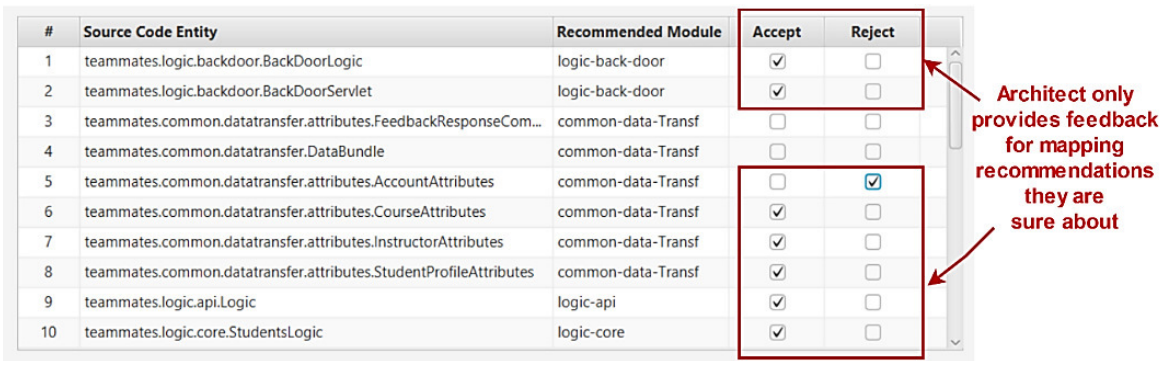
# of **required recommendations in feedback** = # of recommendations on page

How Complete-Batch Mapping Recommendations in InMap Works: Recall that in step *six*, InMap derives the highest scoring class-to-module pairs, from the class-to-module similarity scores (SS_{cm}) matrix, or package-to-module pairs, from the package-to-module similarity scores (SS_{pm}) and gives them as class/package-to-module mapping recommendations. InMap presents, as recommendations, either the class/package-module pairs above the arithmetic mean of the highest similarity scores obtained for a pair; or the best 30 recommendations (if those above the mean are greater than 30). Thirty gave the most optimal results based on the systems tested. In step *seven*, this final filtered list is presented to the architect to review the recommendations given. An architect gives feedback on the page (batch of 30) recommendations produced. In its original form, the architect's feedback provided to InMap must be complete. The architect is expected to provide feedback (an accept/reject) for every recommendation listed on the page.


Unmapped Code
Recommended Architectural Module
Architect must provide feedback for each mapping recommendation produced

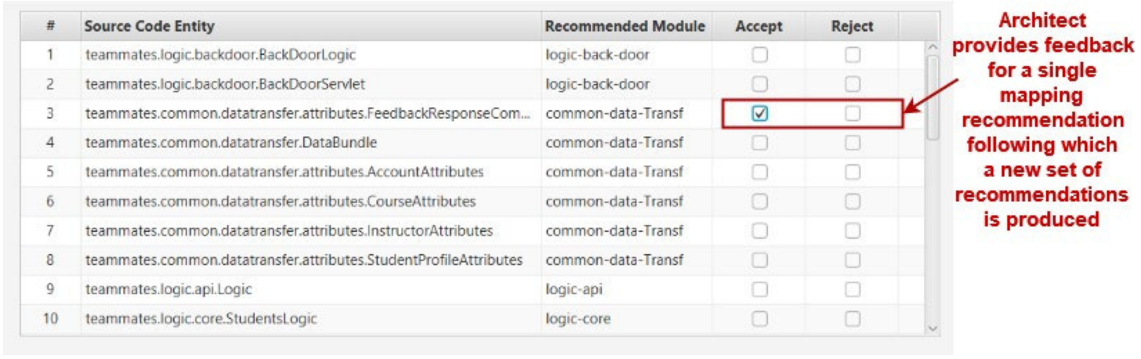
#	Source Code Entity	Recommended Module	Accept	Reject
1	teammates.logic.backdoor.BackDoorLogic	logic-back-door	✓	☐
2	teammates.logic.backdoor.BackDoorServlet	logic-back-door	✓	☐
3	teammates.common.datatransfer.attributes.FeedbackResponseCom...	common-data-Transf	☐	✓
4	teammates.common.datatransfer.DataBundle	common-data-Transf	☐	✓
5	teammates.common.datatransfer.attributes.AccountAttributes	common-data-Transf	✓	☐
6	teammates.common.datatransfer.attributes.CourseAttributes	common-data-Transf	✓	☐
7	teammates.common.datatransfer.attributes.InstructorAttributes	common-data-Transf	✓	☐
8	teammates.common.datatransfer.attributes.StudentProfileAttributes	common-data-Transf	✓	☐
9	teammates.logic.api.Logic	logic-api	✓	☐
10	teammates.logic.core.StudentsLogic	logic-core	✓	☐

Fig 1. Illustration of *complete-batch* feedback. For this approach of collecting feedback, an architect must provide feedback for each mapping recommendation produced.


Architect only provides feedback for mapping recommendations they are sure about

#	Source Code Entity	Recommended Module	Accept	Reject
1	teammates.logic.backdoor.BackDoorLogic	logic-back-door	✓	☐
2	teammates.logic.backdoor.BackDoorServlet	logic-back-door	✓	☐
3	teammates.common.datatransfer.attributes.FeedbackResponseCom...	common-data-Transf	☐	☐
4	teammates.common.datatransfer.DataBundle	common-data-Transf	☐	☐
5	teammates.common.datatransfer.attributes.AccountAttributes	common-data-Transf	☐	✓
6	teammates.common.datatransfer.attributes.CourseAttributes	common-data-Transf	✓	☐
7	teammates.common.datatransfer.attributes.InstructorAttributes	common-data-Transf	✓	☐
8	teammates.common.datatransfer.attributes.StudentProfileAttributes	common-data-Transf	✓	☐
9	teammates.logic.api.Logic	logic-api	✓	☐
10	teammates.logic.core.StudentsLogic	logic-core	✓	☐

Fig 2. Illustration of *partial-batch* feedback. For this approach of collecting feedback, an architect can choose which mapping recommendations they want to provide feedback for and can opt to delay or defer deciding on some.


Architect provides feedback for a single mapping recommendation following which a new set of recommendations is produced

#	Source Code Entity	Recommended Module	Accept	Reject
1	teammates.logic.backdoor.BackDoorLogic	logic-back-door	☐	☐
2	teammates.logic.backdoor.BackDoorServlet	logic-back-door	☐	☐
3	teammates.common.datatransfer.attributes.FeedbackResponseCom...	common-data-Transf	✓	☐
4	teammates.common.datatransfer.DataBundle	common-data-Transf	☐	☐
5	teammates.common.datatransfer.attributes.AccountAttributes	common-data-Transf	☐	☐
6	teammates.common.datatransfer.attributes.CourseAttributes	common-data-Transf	☐	☐
7	teammates.common.datatransfer.attributes.InstructorAttributes	common-data-Transf	☐	☐
8	teammates.common.datatransfer.attributes.StudentProfileAttributes	common-data-Transf	☐	☐
9	teammates.logic.api.Logic	logic-api	☐	☐
10	teammates.logic.core.StudentsLogic	logic-core	☐	☐

Fig 3. Illustration of *real-time* feedback. For this approach of collecting feedback, an architect provides feedback for a single mapping recommendation, following which a new set of recommendations is instantly produced and presented.

B. Partial-Batch Feedback

Partial-batch feedback is an alternative approach to collecting feedback about code-to-architecture mapping recommendations. In this form, an architect provides feedback for a subset of the recommendations provided on a given page. This could be because the architect needs to gain sufficient knowledge about some of the code entities (classes/packages) listed in the recommendations provided and would prefer to delay/defer decisions about those entities. The mapping algorithm would use the partial feedback to provide a new set/page of mapping recommendations that may or may not include the entities the architect did not decide on in prior iterations. *Fig. 2* illustrates this. This implies:

1 < # of **required recommendations in feedback** < # of recommendations on page

How Partial-Batch Mapping Recommendations in InMap Works: In step *seven* of InMap's algorithm, an architect is presented with 30 mapping recommendations to review. The architect does not have to give feedback for all 30 recommendations produced and may opt to skip some, essentially delaying or deferring a decision about them. We say delay or defer as these recommendations could reappear, given that they were not outrightly rejected. However, it is also possible that they may not reappear, given that InMap uses the architect's feedback provided to decide the next set of mapping recommendations to produce.

C. Real-Time Feedback

Real-time feedback is another alternative approach to collecting feedback from the software architect about code-to-architecture mapping recommendations. In this form, the mapping algorithm would produce a new set/page of recommendations immediately after an architect provides feedback on any of the recommendations on the page. *Fig. 3* illustrates this. This implies:

# of **required recommendations in feedback** = 1

How Real-Time Mapping Recommendations in InMap Works: In this case, despite InMap providing 30 mapping recommendations, the list of recommendations provided gets updated immediately after the architect provides feedback on a single code entity, i.e. in real-time. Note that it would be ideal to implement this in a way that the position in the ordered set did not matter, implying an architect can give feedback on any individual code entity from anywhere in the list, and the mapping algorithm refreshes the recommendations instantaneously. However, it is important to acknowledge that recommendation results are always presented with the best candidate at the top or beginning of the list and the worst at the bottom or end of the list. Therefore, in both the *partial-batch* and *real-time* feedback approaches, we consider the rank of the recommendation.

V. EVALUATION

A. Experimentation

To test the effect of our two proposed alternatives to collecting feedback, we ran experiments on InMap in its original form, i.e. *complete-batch feedback* as our control. We then ran the same set of tests on our *partial-batch feedback* approach and our *real-time feedback* approach. We extended the InMap evaluator tool developed in prior studies of InMap [18, 19, 20, 21, 22] to accommodate the evaluation of our two proposed feedback approaches. The evaluation tool simulates a "human architect" accepting and rejecting the recommendations produced. It uses the oracle class/package mappings provided by knowledge experts of each system, as reported in prior studies of InMap [19, 21, 22]. We used the optimal parameter settings for both InMap's class-based version [18, 19] and InMap's hierarchical package-based version [20, 21], observing what effect both *partial-batch* and *real-time feedback* have on both the class-based and package-based versions of InMap. A batch size of 30 was used for the control experiment, i.e. the *complete-batch feedback*. However, for the *partial-batch feedback,* we tested a range of batch sizes, in addition to a batch size of 30, to observe if different batch sizes affect the results.

For every experiment, we collected the *recall*, *precision* and *f_1-scores* (as a harmonic mean between recall and precision) of the recommendations produced. InMap produces mapping recommendations in descending order of the most likely correct recommendation based on similarity scores. We take it as a norm that as an architect reviews a list of recommendations provided, they start reviewing the list and making decisions in sequential order from the beginning/top to the end/bottom instead of reviewing it randomly. This approach is similar to how most other recommendation or information retrieval-based systems are designed. They surmise that the best candidate in a list of results is found at the beginning/top of the list and the worst candidate at the end/bottom of the list. Therefore, in reviewing both *partial-batch* and *real-time feedback,* we consider the rank of a recommendation.

B. Systems Under Test

We evaluated our modified versions of InMap (*partial-batch* and *real-time feedback*) against InMap in its original form (*complete-batch feedback*) using six Java-based open-source systems used in prior InMap studies. These are *Ant*, a command line and API tool for automating processes; *ArgoUML*, a desktop application for modelling in UML; *JabRef,* a desktop application for managing bibliographic references; *Jittac,* an Eclipse IDE plugin for applying reflexion modelling; *ProM,* a desktop application for mining processes; and *TeamMates* a web application for peer reviews and feedback. These systems all have varying characteristics in terms of the number of lines of code, number of architectural modules, length of architectural descriptions, number of source files, number of classes and number of packages, to name a few. Our prior studies documented their characteristics [18, 19, 20, 21, 22].

C. Replication Package

A replication package of the evaluation tool; the six case systems tested along with their independently produced ground-truth mappings provided by experts knowledgeable about the respective systems; and the complete, partial and real-time feedback mapping approaches and results are all available in the online open-repository at the following link https://doi.org/10.6084/m9.figshare.13714150.

VI. Results

Tables 1, 2 and *3* show the results obtained for both InMap's class-based and package-based algorithms when run using the six test-case systems. They show the recall, precision and f_1-score for each version of the algorithm checked against each feedback approach. Green denotes an increase, whereas red denotes a decrease.

Table 1 gives the results obtained for the control experiment, i.e. the *complete-batch feedback* approach, which is how InMap was initially designed to collect feedback from an architect. The average recall for the six systems tested on the class-based version of InMap was 0.972; the average precision was 0.788, and the average f_1-score of 0.870. The package-based version had an average recall of 0.865, average precision of 0.745 and an average f_1-score of 0.800.

Tables 2 and *3* show the results of the two new approaches, i.e*., partial-batch* and *real-time feedback*. The results show that as far as the recall is concerned, these two ways of collecting feedback seem not to affect the recall – both approaches maintained an average of 0.972 for the class-based version and 0.865 for the package-based version of InMap.

With regard to the precision, we see some variances, albeit minor. For the class-based version of InMap, ProM's precision improved in both approaches, with the most significant result coming from the *real-time feedback* approach (3% increase). However, whereas *partial-based feedback* was the same for Ant compared to *complete-batch,* it surprisingly reduced by 1% for the real-time-based approach. Jittac improved in both approaches, with *real-time feedback* recording a higher increase in precision (5%) between the two approaches. ProM also increased precision for the package-based version using the *real-time feedback* approach. Ant again recorded a slight reduction in this case. It was interesting to observe that InMap's package-based version had more movement in precision compared to the class-based version.

The average f_1-scores of all three techniques show that *partial feedback* did not negatively affect the results compared to *complete feedback.* Furthermore, the f_1-scores show that *real-time feedback* gave the best results for all three approaches to collecting feedback.

VII. Discussion

A. Findings

The results of our investigation show that if we update the list of recommendations provided in real-time, the precision of the recommendations improves on average. This is likely because InMap uses the feedback given by an architect in deciding the next set of mapping recommendations to give. In other words, it benefits from getting information early in the mapping process. When we have a batch size of, say, 30 for both the batch processes, that is, *complete* and *partial*, the architect gives feedback for a minimum of 2 and a maximum equal to the batch/page size, in this case, 30. Now, if we consider that each class/package contains a unit of information that InMap could use to make a more accurate mapping recommendation, then in the batch process, we are delaying the feedback loop. The larger our batch/page size, the more significant the delay in relaying what could otherwise be helpful information in predicting the unmapped source entities. In other words, the results show that even though InMap uses information from the architect's feedback to work out the most suitable recommendations, it does not necessarily benefit from having lots of information given to it at once, say feedback on 100 classes in one go. Instead, having smaller units of information fed into the mapping loop early on is more beneficial. Thus, SACC tool developers and architects are more likely to benefit from using a *real-time feedback* approach. However, although *real-time feedback* offers the best results, the difference between both batch processes was shown to be minor. Therefore, if a tool developer or the users of the tools prefer not to work in real-time but give feedback a batch at a time, that works reasonably well too.

The results also show that for the batch feedback mapping approaches (i.e. *complete feedback* and *partial feedback*), on average, if an architect opts to delay decisions about some of the recommendations provided in a list, the recall and precision of the recommendations InMap provides are not negatively affected. Meaning given a batch size of 30, whether an architect chooses to provide feedback on all 30, i.e. *complete feedback*, or whether an architect decides to provide feedback for at minimum 2 or out of the 30 while delaying the rest, maybe because the architect is unsure and would like to see more/other recommendations first, this would not affect the results negatively. On the contrary, the results showed a slight improvement. This could be attributed to the same reasons that *real-time feedback* showed the best results. When an architect gives *partial feedback*, this is a smaller chunk of information than the batch size. Moreover, since InMap has shown that it benefits from receiving feedback as early as possible, giving feedback, for example, for 8 out of 30 recommendations, provides a smaller chunk of information earlier in the feedback loop than providing recommendations for all 30 at once. However, again, in this case, it does not imply that if a tool developer or architect opts for a *complete batch feedback* approach, then the accuracy of the recommendations will reduce drastically. On the contrary, the results showed *partial-batch feedback* recorded a slight increase over *complete-batch feedback*, which already had some reasonably good results.

We must note that there is a limit to the complete-batch size that can be or should be used. Firstly, if we set the batch size to 50 or 100, it is not practical to make an architect provide complete feedback for such a large quantity before providing

TABLE I.
RESULTS OF *COMPLETE-BATCH* FEEDBACK

System	**Class Recall**	**Class Precision**	**Class F_1 Score**	**Package Recall**	**Package Precision**	**Package F_1 Score**
AT	1.00	0.70	0.82	0.77	0.89	0.82
AU	0.99	0.75	0.85	0.78	0.56	0.65
JR	0.99	0.95	0.97	0.95	0.87	0.91
JT	0.99	0.80	0.88	0.82	0.58	0.68
PM	0.98	0.58	0.73	0.87	0.65	0.74
TM	0.88	0.95	0.91	1.00	0.92	0.96
Avg	**0.972**	**0.788**	**0.870**	**0.865**	**0.745**	**0.800**

TABLE II.
RESULTS OF *PARTIAL-BATCH* FEEDBACK

System	**Class Recall**	**Class Precision**	**Class F_1 Score**	**Package Recall**	**Package Precision**	**Package F_1 Score**
AT	1.00	0.70	0.82	0.77	0.89	0.82
AU	0.99	0.75	0.85	0.78	0.56	0.65
JR	0.99	0.95	0.97	0.95	0.87	0.91
JT	0.99	0.80	0.88	0.82	**0.61**	**0.70**
PM	0.98	**0.59**	**0.74**	0.87	0.65	0.74
TM	0.88	0.95	0.91	1.00	0.92	0.96
Avg	***0.972***	***0.790***	***0.872***	***0.865***	***0.750***	***0.803***

TABLE III.
RESULTS OF *REAL-TIME* FEEDBACK

System	**Class Recall**	**Class Precision**	**Class F_1 Score**	**Package Recall**	**Package Precision**	**Package F_1 Score**
AT	1.00	**0.69**	0.82	0.77	**0.87**	0.82
AU	0.99	0.75	0.85	0.78	0.56	0.65
JR	0.99	0.95	0.97	0.95	**0.86**	0.90
JT	0.99	0.80	0.88	0.82	**0.63**	0.71
PM	0.98	**0.61**	**0.75**	0.87	**0.68**	0.72
TM	0.88	0.95	0.91	1.00	0.92	0.96
Avg	***0.972***	***0.792***	***0.873***	***0.865***	***0.753***	***0.805***

a new set of recommendations of similar size. It is tedious to do so for such a large number at a time and does not help reduce the effort required by an architect, which is a core motivation for automating the mapping processes. Secondly, this study and prior studies [18, 19] have shown that smaller units of information fed back into the algorithm at a time improve the results. So whereas the results might be similar for a batch size of 20, 30 or 40, the same cannot be said for sizes of 50, 100 or 150. Generally speaking, a batch size of 30 was shown to produce the best average results in our previous studies, and the larger the batch size is beyond 30, the less our precision becomes, on average.

Both findings on *real-time* and *partial-batch feedback* show that InMap allows flexibility in how a SACC tool collects mapping recommendation feedback because all three approaches have, on average, f_1-scores within 0.005 of each other. Furthermore, if the developer of a SACC tool decides to implement all three approaches, then this flexibility extends to the architect. They can choose their preferred way of providing feedback, i.e. *complete-batch*, *partial-batch* or *real-time.* Furthermore, they can alternate among these approaches throughout the mapping process without committing to a singular approach.

Interestingly, the package-based version of InMap showed better improvement in the average of the f_1-score for both *partial* and *real-time feedback* over the class-based version, 0.005 versus 0.003, respectively. However, the class-based version of the InMap achieved higher f_1-scores for all approaches compared to the package-based version, 0.87 vs 0.80; therefore, there was less opportunity for improvement for the class-based version compared to the package-based version because it already had reasonably high-scoring results. That said, an approach combining both the class-based and package-based versions of InMap, such as the one introduced in [22], would likely derive improvements from both versions.

B. Limitations & Validity

The two new feedback approaches introduced build on top of InMap's *class* and *package similarity* functions; therefore, the same factors that affect the external validity of InMap's results are inherited by the alternative feedback approaches presented in this paper. That is to say, aspects such as the code commenting quality and style; the number of classes, packages and modules; and the length and quality of the architecture description could likely affect the external validity of our results. Therefore, more case systems with variable attributes would add to the soundness of the results. Nonetheless, the results of the six systems tested and their varying characteristics provide a fair case for the two feedback approaches investigated.

VIII. Conclusion & Future Work

This paper presents two alternative approaches to how an architect can provide feedback on mapping recommendations provided, either as *partial-batch* or *real-time feedback*. They are an alternative to InMap's *complete-batch feedback* approach. Our *partial-batch feedback* approach showed a net increase of 2% in precision for the systems tested, and our *real-time feedback* approach showed a net increase of 5% in precision for the systems tested. This shows that providing *partial–batch feedback* does not harm the precision of the recommendation produced compared to when *complete-batch feedback* is provided. Furthermore, it shows that providing feedback in *real-time* improves the recommendations produced. Moreover, because the results for *complete-batch feedback* were already reasonably good, the 2-5% increase that these two new approaches provide allows for flexibility in how SACC tools that use *human-in-the-loop* approaches can collect feedback; or flexibility in how architects themselves opt to provide feedback. This implies that tools that use InMap's automated interactive mapping algorithm have some flexibility in how they choose to gather feedback from an architect as mapping progresses (*complete* vs *partial* and *batch* vs *real-time*) without suffering a loss in recall and with an insignificant difference in precision. It also offers flexibility for an architect, assuming a SACC tool implements the different ways mapping recommendation feedback can be collected.

In future work, we would like to do more detailed studies with more systems to further test the soundness of the conclusion of this study. We would also like to see how the two new feedback approaches introduced in this work would fair on the version of InMap that integrates both the class-based and package-based versions of the algorithm into one [22]. Lastly, we would like to carry out an exploratory study that examines the cases that are difficult for InMap to map with either mapping version of InMap or either approach to collecting feedback.

References

[1] N. Ali et al, "Architecture Consistency: State of the Practice, Challenges and Requirements," in *Empirical Software Engineering*, 23(1), 2018, pp. 224–258, https://doi.org/10.1007/s10664-017-9542-0

[2] M. Bauer, M. Trifu, "Architecture-Aware Adaptive Clustering of OO Systems," Proceedings – 8th European Conference on Software Maintenance and Reengineering, 2004, pp. 3–14, https://doi.org/10.1109/CSMR.2004.1281401

[3] R.A. Bittencourt et al, "Improving Automated Mapping in Reflexion Models Using Information Retrieval Techniques," *Proceedings – Working Conference on Reverse Engineering, WCRE*, 2010, pp. 63–172, http://dx.doi.org/10.1109/WCRE.2010.26

[4] A. Christl et al, "Automated Clustering to Support the Reflexion Method," in *Information and Software Technology*, 49(3), 2007, pp. 255–274, https://doi.org/10.1016/j.infsof.2006.10.015

[5] A. Christl et al, "Equipping the Reflexion Method with Automated Clustering," *12th Working Conference on Reverse Engineering*, 2005, https://doi.org/10.1109/WCRE.2005.17

[6] F.A. Fontana et al, "Tool Support for Evaluating Architectural Debt of an Existing System: An Experience Report," *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 2016, pp. 1347–1349, http://dx.doi.org/10.1145/2851613.2851963

[7] N. Medvidovic, R.N. Taylor, "Software Architecture: Foundations, Theory, and Practice", *ACM/IEEE 32nd International Conference on Software Engineering*, 2010, pp. 471–472, https://doi.org/10.1145/1810295.1810435

[8] J. Knodel, "Sustainable Structures in Software Implementations by Live Compliance Checking," *Fraunhofer-Verl*, Stuttgart, 2011.

[9] J. Knodel, D. Popescu, "A Comparison of Static Architecture Compliance Checking Approaches," *Proceedings of the 6th Working IEEE/IFIP Conference on Software Architecture*, 2007, https://doi.org/10.1109/WICSA.2007.1

[10] G.C. Murphy et al, "Software Reflexion Models: Bridging the Gap between Source and High-Level Models," *IEEE Transactions on Software Engineering*, 27(4), 2001, pp. 364–380, https://doi.org/10.1109/32.917525

[11] S.M. Naim et al, "Reconstructing and Evolving Software Architectures Using a Coordinated Clustering Framework", in *Automated Software Engineering*, 24(3), 2017, pp. 543–572, https://doi.org/10.1007/s10515-017-0211-8

[12] T. Olsson et al, "Semi-Automatic Mapping of Source Code using Naive Bayes," *Proceedings of the 13th European Conference on Software*

Architecture ECSA, *Lecture Notes in Computer Science*, 13365, 2022, pp. 65-85, https://doi.org/10.1007/978-3-031-15116-3_4

[13] L. Passos et al, "Static Architecture-Conformance Checking: An Illustrative Overview," in *IEEE Software*, 2010, 27(5), pp. 82–89, https://doi.org/10.1109/MS.2009.117

[14] D.E. Perry, A.L. Wolf, "Foundations for the Study of Software Architecture," in *SIGSOFT Softw. Eng. Notes*. 17, 4, 1992, pp. 40–5, https://doi.org/10.1145/141874.141884

[15] J. Rosik et al, "Assessing Architectural Drift in Commercial Software Development: A Case Study," in *Software Practice and Experience*, 41, 2011, pp. 63–86, https://doi.org/10.1002/spe.999

[16] L. de Silva, D. Balasubramaniam, "Controlling Software Architecture Erosion: A Survey," in *Journal of Systems and Software*, 85(1), 2012, pp. 132–151, https://doi.org/10.1016/j.jss.2011.07.036

[17] Z.T. Sinkala, S. Herold, "InMap: Automated Interactive Code-to-Architecture Mapping," *Proceedings of the ACM Symposium on Applied Computing*, 2021, pp. 1439–1442, https://doi.org/10.1145/3412841. 3442124

[18] Z.T. Sinkala, S. Herold, "InMap: Automated Interactive Code-to-Architecture Mapping Recommendations," *Proceedings – IEEE 18th International Conference on Software Architecture*, 2021, pp. 173–183, https://doi.org/10.1109/ICSA51549.2021.00024

[19] Z.T. Sinkala, S. Herold, "Towards Hierarchical Code-to-Architecture Mapping Using Information Retrieval," *Companion Proceedings – IEEE 15th European Conference on Software Architecture,* 2021.

[20] Z.T. Sinkala, S. Herold, "Hierarchical Code-to-Architecture Mapping," in *ECSA 2021 Tracks and Workshops – Revised Selected Papers*, 2022, https://doi.org/10.1007/978-3-031-15116-3_5

[21] Z.T. Sinkala, S. Herold, "An Integrated Approach to Package and Class Code-to-Architecture Mapping Using InMap," *Proceedings – IEEE 20th International Conference on Software Architecture*, 2023, https://doi.org/10.1109/ICSA56044.2023.00023

[22] T.A. Wiggerts, "Using Clustering Algorithms in Legacy Systems Remodularization," *Proceedings of the 4th Working Conference on Reverse Engineering*, 1997, pp. 33–43, https://doi.org/10.1109/WCRE.1997.624574