A Preliminary Study on the Use of Keywords for Source Code to Architecture Mappings

Tobias Olsson, Morgan Ericsson and Anna Wingkvist

Department of Computer Science and Media Technology, Linnaeus University, Kalmar/Växjö, Sweden

Abstract
We implement an automatic mapper that can find the corresponding architectural module for a source code file. The mapper is based on multinomial naive Bayes, and it is trained using custom keywords for each architectural module. For prediction, the mapper uses the path and file name of source code elements. We find that the needed keywords often match the module names, but also that ambiguities and discrepancies exist. We evaluate the mapper using nine open-source systems and find that the mapper can successfully create a mapping with perfect precision, but in most cases, it cannot cover all source code elements. Other techniques can, however, use the mapping as a foothold and create further mappings.

Keywords
Orphan Adoption, Software Architecture, Source Code Clustering, Naive Bayes

1. Introduction

The modular software architecture captures major design decisions regarding reuse, maintainability, changeability, and portability [1]. During system evolution, the source code must conform to the architecture, or the system risks accumulating technical debt and finally lose the desired qualities.

Static Architecture Conformance Checking (SACC) methods, such as Reflexion modeling [2], statically analyze source code to ensure that it does not introduce architectural violations [3, 4]. These methods require an architecture model, with modules and dependencies, and a source code model, with entities (e.g., source code files) and concrete dependencies (e.g., due to inheritance or method invocations). They also require a mapping from the source code model to the architecture model to detect convergent, absent, or divergent dependencies in the implementation.

Despite the importance of architecture conformance, SACC has not reached widespread use in the software industry [1, 3, 5, 6]. The necessary tools and methods for using SACC exist. However, practitioners perceive the mapping from source code to architectural modules as a significant hindrance; it is often outdated or nonexistent. Many tools address this by combining manual mapping and regular expressions to filter file, module, and package names. Still, such are considered to be both time-consuming and error-prone [3, 5, 6, 7].

Automatic mapping techniques aim to minimize the manual effort needed to create a mapping by using information available in the source code and intended modular architecture. For example, dependencies between source code entities can be used to create a mapping. A problem with current automatic techniques is that they require an initial set of mapped entities that the technique infers the automatic mappings from. Depending on the technique and system to be mapped, an initial set needs to consists of approximately 15-20% of the entities before reaching acceptable performance. In our experience, the physical structure of files on disk is often in part or wholly reflected in the intended modular architecture. Effective use of this information can present an attractive option to create an initial set. However, structure and naming are not always mapped one to one to a module, and there are discrepancies, ambiguities, or simply missing terms in the naming.

We investigate how well a multinomial naive Bayes classifier trained using simple keywords derived from ground truth mappings can be used to automatically create an initial set. We pose the following questions:

1. Can the mapper construct an initial set based on a simple set of keywords for each module?
2. How well does this initial set perform if used in combination with mapping based on dependencies?
3. How well does the above combination perform compared to the NBAttract (with a random initial set) and InMap approaches?

We evaluate the mapper using nine open-source systems with known mappings to a specified modular architecture and find that the keywords are often the same as the module names, but more and different keywords are needed in some cases. After the initial set is created, we run another automatic mapper that can map

ECSA2021 Companion Volume
tobias.olsson@lnu.se (T. Olsson); morgan.ericsson@lnu.se (M. Ericsson); anna.wingkvist@lnu.se (A. Wingkvist)
0000-0003-1154-5308 (T. Olsson); 0000-0003-1173-5187 (M. Ericsson); 0000-0002-0835-823X (A. Wingkvist)

any remaining entities. We compare the results with a traditional automatic mapping technique [8] and an interactive mapping technique [7]. We find that the keywords-based approach can, in some cases, provide a complete mapping and that the keywords-based approach plus the automatic mapping approach performs very well.

2. Background and Related Work

Tzerpos and Holt describe the general problem of mapping (or remapping) a source code entity to an architectural module [9]. They collectively call both the mapping and remapping of an entity the orphan adoption problem. They find four major criteria for solving the problem: *naming*, *structure*, *style*, and *semantics* and device an algorithm that they evaluate in three case studies [9]. Tzerpos and Holt regard the naming criteria as the first option to use in an orphan adoption scenario and suggest using per system regular expressions to determine a mapping. However, they also mention that naming criteria is not enough as they may be lacking or that naming standard is not always adhered to by developers.

Garcia et al. discuss the use of package and naming information in software architecture recovery [10]. In general, they found that their ground truth components often spanned or shared several packages. They could not find a correlation between components and single package or directory names. One of their four cases presented a reasonably good correlation, and in one system, they could find a repeating pattern of directories. The ground truth architectures recovered in their study are possibly at a lower level than the modular architectures we study. Still, there is likely variation in what dimension or view of an architecture is expressed in the package structure. This variation is further supported by Buckley et al., where one out of five studied systems did not have any clear correlation between packages and modules. This presented difficulties and significant effort when performing manual mapping [11].

Anquetil and Lethbridge, on the other hand, propose a method for architecture recovery of legacy systems using filenames [12]. Their approach focuses on the assumptions that files have short names with many abbreviations and are placed in a single directory. This is due to their focus on recovering legacy systems. Nevertheless, they present some interesting findings. First, they identify several forces that shape a filename, i.e., what influences it. There seem to be several examples of such forces also in more modern implementations, e.g., from the subject system *Ant*, we find the feature implemented (*ant.taskdefs.SendEmail*), the algorithms or steps of algorithms (*ant.types.resources.Sort*), or data processed (*ant.taskdefs.email.Header*), as suggested in [12]. Much of the approach revolves around the problematic abbreviations found in the relatively short filenames. While this is not a technical problem in modern development, the use of abbreviations is still common practice. For example, one of the subject systems, *ArgoUml*, defines a module *reverseEngineering*, and the corresponding directory mapping is the abbreviation *reveng*. Finally, Anquetil and Lethbridge successfully use filenames to create a clustering that corresponds well to an expert's view of a system.

2.1. Semi-Automatic Mapping

Christl et al. introduced the Human Guided clustering Method (HuGMe), an approach to semi-automatic mapping of source code entities to modules of the intended architecture [9]. It is an iterative approach that, at its core, uses an attraction function to compute the attraction between a source code entity and a module. If the attraction is considered valid, an automatic mapping is made; if not, the attractions can be used as a suggestion for a human user. Two attraction functions based on dependencies are presented, CountAttract and MQAttract [13, 6].

Bittencourt et al. present two new attraction functions based on information retrieval techniques [5]. They use semantic information in the source code, including module- and filenames. The attractions are calculated based on cosine similarity (IRAttract) and latent semantic indexing (LSIAttract). They make a quantitative comparison between the performance of their attraction functions with CountAttract and MQAttract in an evolutionary setting (where a few new files are to be assigned a mapping). They find that combining attraction functions (e.g., if CountAttract fails, try IRAttract) performs best. They find that CountAttract usually misplaces entities on module borders. MQAttract performs better when mapping entities with dependencies to many different modules. IRAttract and LSIAttract perform better when mapping entities in libraries or entities on module borders, but worse if there are modules that share vocabulary but are not related [5].

We have created an attraction function that uses machine learning techniques and introduced the Concrete Dependency Abstraction (CDA) method [8]. In short, CDA produces textual representations of dependencies at the level of architectural modules and lets a machine learning technique learn the patterns of dependencies from the actual source code and combine these with information retrieval techniques. We implement this approach using naive Bayes as an attraction function for the HuGMe method, NBAttract. We have compared the automatic mapping performance of CountAttract, IRAttract, LSIAttract and NBAttract over several systems using *s4rdm3x*, our open-source tool suite for automatic mapping experiments [8, 14].

The main limitations for the techniques that build on HuGMe are the need for an initial set and, in some cases, low-quality mappings. The initial set needs to be manually created and be of good quality for the attraction functions to perform well. We estimate that a randomly composed initial set needs to include approximately 15-20% of the source code entities. Based on this, we conclude that creating the initial set is likely a significant effort. Automated techniques will probably not result in a perfect mapping except when they use a large initial set and only map a few entities. In the best of cases, the automated technique leaves hard to map instances to the user (creating more manual work), but misclassifications are problematic. There has not been much research in the manual mapping steps of HuGMe except for the original studies [13, 6]. Handling of misclassification and manual support in these methods are still open issues.

2.2. Interactive Mapping

Sinkala and Herold present InMap, which is not an automated approach to mapping per se, but instead suggest mappings to the end-user, who can then choose to accept the suggested mapping (or not) [7]. It is an iterative approach that iteratively presents a suggested mapping for a fixed number of entities. The end-user chooses to accept or reject the suggestions. InMap uses the accepted mappings to improve the suggested mappings further in the next iteration. It also uses the negative evidence of a rejected mapping and does not suggest this mapping again. InMap produces the suggested mappings similar to Bittencourt et al., with the addition of a descriptive text for each architectural module. InMap also includes the path and filename used in the Java class and package names. It treats the source code entities as a database of documents and uses Lucene to search this database using module information as a query. Sinkala and Herold evaluate InMap using six open source systems. For the best combination (in terms of highest F1 score) of information, InMap can suggest mappings for most of a system's entities with a mean recall of 0.95, a mean precision of 0.84, and a mean F1 score of 0.89.

The main limitations of InMap are its highly interactive nature and that architectural documentation needs to exist for every module. The documentation provided needs to be of good quality, i.e., as short as possible but containing good keywords. Noisy documentation will likely not help in producing high-precision suggestions. The interactiveness of InMap is in some way double-edged; the technique often seems to require more interaction (accepting or rejecting a suggested mapping) than there are entities in the source code. On the other hand, if not minor mapping errors can be tolerated, a mapping validation is needed anyway.

3. Keywords and File-Based Mapping

File naming and structure seem to reflect the intended modular architectures we have studied quite well. For example, module names tend to map to the directory structure of the source code. However, the naming is often not perfect. In some cases, module names are not used, or shorter or slightly different terms are used. In other cases, several module names exist in the structure or naming of a file. A simplistic approach is thus not appropriate. Instead, the file naming patterns need to be fully defined, e.g., using regular expressions or a heuristic. For regular expressions to work, there is often a need to maintain several expressions that can be conflicting and overlapping. A more attractive option would be to use machine learning and train a classifier using a good set of keywords. The classifier's task is to produce a good enough initial set. An automatic mapping technique can then use this initial set for further mappings.

In this work, we implement a proof of concept mapper using a multinomial naive Bayes classifier. It is a simple, probabilistic approach that uses word frequencies to compute the probability of each class. While it is conceptually simple, naive Bayes often produce good results, especially if the training data is small. As the goal is to create a good enough mapping using a small set of predefined keywords, naive Bayes is thus a good candidate for a proof of concept study.

We base our implementation on the Weka library [15] and train the classifier using the custom keywords for each module. Note that the same keyword can be specified multiple times, increasing the importance of that particular keyword.

We derive the prediction data from the path of each source code entity, including the filename. The filename is split into words based on common camel-, kebab, and snake-case rules. In addition, we value later parts of the path more and add these words multiple times. Intuitively allowing for a deeper nested folder mapping to "override" a higher level mapping. For example, the file:

net/sf/jabref/logic/util/io/FileHistory.java

will produce the following words:

net sf jabref logic util io filehistory file history sf jabref logic util io jabref logic util io logic util io util io io

Note the six occurrences of *io* reflecting the nesting depth of the word in the path.

To generate a useful initial set, it is more important that the mappings are precise rather than complete. There needs to be a high difference between the best mapping probability and the second best. By trial and error, we found a factor of 1.99 to work well, i.e., the highest probability needs to be 1.99 times higher than the second-highest probability for mapping to occur.

We have implemented the mapper described above in our open-source tool suite *s4rdm3x* [16].

4. Method

We use nine open-source systems where the ground truth mappings are known. We create a keyword set for each module based on the ground truth mappings. We make sure that these keywords will successfully map at least some entities to each module.

After we have determined the keywords, we run our keywords-based mapper and create an initial set. This initial set is then used as the input to another mapper, NBAttract, which also uses multinomial naive Bayes but instead forms training- and prediction words using dependency information in the form of concrete dependency abstractions (CDA) [8]. We compare the performance to NBAttract with a random initial set. In this configuration, we use file information (not including the module keywords) and CDA. In addition, we compare to the interactive approach InMap [7].

We collect precision, recall, and combined F1 scores for each approach. When a random initial set is used, several sets of different sizes and compositions are needed to cover a large range of combinations. We will present the performance metrics numerically and visually as the effect of the initial set size is essential.

We use nine open-source systems implemented in Java. Ant[1] is an API and command-line tool for process automation. ArgoUML[2] is a desktop application for UML modeling. Jabref[3] is a desktop application for managing bibliographical references. K9[4] is an open-source email client for Android. Lucene[5] is an indexing and search library. ProM[6] is an extensible framework that supports a variety of process mining techniques. Note that we use the ProM framework and not the full ProM system. Sweet Home 3D[7] is an interior design application. TeamMates[8] is a web application for handling student peer reviews and feedback.

A documented software architecture and a mapping from the implementation to this architecture exist for each system. Jabref, TeamMates, and ProM have been the study subjects at the Software Architecture Erosion and Architectural Consistency Workshop (SAEroCon) 2016, 2017, and 2019 respectively. A system expert has provided both the architecture and the mapping for these systems. The architecture documentation and mappings are available in the SAEroCon repository[9]. ArgoUML, Ant, and Lucene has been previously studied [17, 18], and the architectures and mappings were extracted from the replication package of Brunet et al. [17]. K9 has been preliminary mapped by ourselves based on architecture documentation provided in [19][10]. We have not validated this mapping with system experts but include it since it is an interesting case with a more complex file structure.

[1]https://ant.apache.org
[2]http://argouml.tigris.org
[3]https://jabref.org
[4]https://k9mail.app/
[5]https://lucene.apache.org
[6]http://www.promtools.org
[7]http://www.sweethome3d.com
[8]https://teammatesv4.appspot.com

5. Results and Analysis

We use the existing ground truth mappings to construct a set of keywords for each system. Table 1 shows the manually extracted keywords. Note that a single keyword is sufficient in many cases, and many keywords are the same as or some variation of the module name. K9 presents an interesting exception where several keywords are needed. We relied on a high-level architectural description when creating the mapping for K9, where allowed dependencies were the most clearly defined. The keywords used reflect the sub-modules of the high-level modules. Note that our mapping has not been validated by systems experts.

Using the generated initial sets, we ran the NBAttract mapper with CDA information only. We ran 1530 experiments with random initial sets for the NBAttract mapper where the mapper used filename and CDA information (no module keywords). Finally, we use the best-reported performance metrics from [7]. Table 2 shows the comparison of the four approaches. Using the keywords-based mapping, we can create an initial set with perfect precision and recall in Commons Imaging, ProM, and Sweet Home 3D. The keywords for these systems are straightforward and are often directly reflected in the module name. For the other systems, keywords can generate an initial set with perfect precision. However, recall is suffering.

Using the keywords-based initial sets and NBAttract using CDA performs very well, with precision scores over 0.95 in all cases and almost perfect scores for recall, cf. Table 2).

Figures 1, 2, and 3 shows the running median F1 score, precision, and recall for each system. The figures focus on showing the running median for random initial sets and NBAttract. This configuration seems to lack precision in Commons Imaging and Sweet Home 3D, and the recall is suffering in Ant. The naming and dependency information are possibly conflicting in these systems. Ta-

[9]https://github.com/sebastianherold/SAEroConRepo
[10]http://oss.models-db.com/Downloads/EASE2019_ReplicationPackage/

Table 1
Keywords for each system and module.

System	Module	Keywords	System	Module	Keywords
Ant	compilers	2 * compiler	JabRef	globals	globals
		2 * compilers		preferences	preferences prefs
	condition	condition		model	model shared dbms
	rmic	rmic		logic	logic shared
	cvslib	cvslib		gui	gui
	email	email		cli	cli
	taskdefs	taskdefs	Lucene	queryparser	queryparser
	listener	listener		search	search
	types	types		index	index
	ant	ant		store	store
	util	util		analysis	analysis
	zip	zip		util	util
	tar	tar		document	document
	mail	mail	K9	business	controller service
	bzip2	bzip2			mail k9 power
AUML	application	2 * application			search migrations
	diagrams	2 * diagram		presentation	activity ui notification
	notation	notation			fragment view list
	explorer	explorer			widget helper crypto
	codeGeneration	3 * language code		service	provider action extra
		generation		dataaccess	mailstore util
	javaCodeGeneration	language code		crosscutting	crypto autocrypt
		generation 2 * java			cache helper
	reverseEngineering	3 * reveng	ProM	framework	framework
	persistence	persistence		contexts	contexts
	moduleLoader	moduleloader 2 * api		models	models
		module modules		plugins	plugins
	gui	ui	SH3D	sH3DModel	model
	model	model		sH3DTools	tools
	internationalization	i18n		sH3DPlugin	plugin
	swingExtensions	swingext		sH3DViewController	viewcontroller
	ocl	ocl		sH3DSwing	swing
	critics	2 * cognitive		sH3DJava3D	j3d
C Img	base	imaging		sH3DIO	io
	color	color		sH3DApplet	applet
	common	common		sH3DApplication	sweethome3d
	bmp	bmp	TMates	common.util	util
	dcx	dcx		common.exception	exception
	gif	gif		common.dataTransfer	datatransfer
	icns	icns		ui.automated	automated
	ico	ico		ui.controller	controller
	jpeg	jpeg		ui.view	ui page
	pcx	pcx		logic.core	core
	png	png		logic.api	logic api
	pnm	pnm		logic.backdoor	backdoor
	psd	psd		storage.entity	entity
	rgbe	rgbe		storage.api	storage api
	tiff	tiff		storage.search	search
	wbmp	wbmp		testDriver	2 * test
	xbm	xbm		client.remoteAPI	remoteapi
	xpm	xpm		client.scripts	2 * scripts
	icc	icc			
	internal	internal			
	palette	palette			

Table 2
Precision, Recall and F1 score for each mapping technique. For Random + NBAttract, the median metrics are shown.

	Keywords			Keywords + NBAttract			Random + NBAttract			InMap		
System	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
Ant	1.00	0.97	0.99	0.99	1.00	0.99	0.94	0.91	0.94	0.73	1.00	0.84
AUML	1.00	0.67	0.80	0.97	1.00	0.98	0.95	1.00	0.97	0.78	0.98	0.87
C Img	1.00	1.00	1.00				0.84	0.99	0.90			
JabRef	1.00	0.95	0.98	0.98	1.00	0.99	0.91	0.98	0.94	0.96	1.00	0.98
K9	1.00	0.81	0.90	0.96	1.00	0.98	0.92	1.00	0.96			
Lucene	1.00	0.99	1.00	1.00	0.99	1.00	0.97	1.00	0.98			
ProM	1.00	1.00	1.00				0.99	1.00	1.00	0.81	0.87	0.84
SH3D	1.00	1.00	1.00				0.83	1.00	0.91			
TMates	1.00	0.60	0.75	0.97	1.00	0.99	0.97	1.00	0.98	0.95	0.97	0.96
Mean	*1.00*	*0.89*	*0.93*	*0.98*	*1.00*	*0.99*	*0.92*	*0.99*	*0.95*	*0.846*	*0.964*	*0.90*

ble 2 shows mean values; they can vary quite a bit in the actual cases depending on the size and composition of the initial set.

Finally, InMap lacks in precision but performs well regarding the recall. Note that InMap is a highly interactive approach to mapping. The aim is not to automate the mapping but rather give good advice to a human user that interactively maps the source code iteratively. If there is a need to check an automatic mapping thoroughly, an interactive approach is attractive regardless of precision.

6. Discussion and Validity

Keywords can be effectively used and provide an excellent initial set, even a perfect mapping in some cases. It is an attractive approach compared to manually mapping an initial set. Hypothetically, it should be easier to extract the keywords and specify the corresponding module and weight of the keyword than mapping several tens or hundreds of files manually. The main challenge in this area is, of course, to find a high precision and minimal set of keywords. We used the already established ground truth mappings to do this in this preliminary evaluation, but this approach is not feasible in a real case. However, analyzing the directory structure and looking for words in the module names could provide a starting point in many cases. Possibly using a deeper level in the directory hierarchy or looking for repeating patterns could be fruitful. Semantic analysis using, e.g., WordNet could be an approach to find related words in the directory structure. In addition, information from, e.g., method names and identifiers could be used.

It would arguably be easier to create and maintain a small set of keywords compared to, e.g., regular expressions, even if done entirely manually.

Using a large random initial set seems to give a very high performance of NBAttract in some cases, e.g., ArgoUML, Commons Imaging, Lucene, ProM, Sweet Home 3D (cf. Figure 1). This indicates that when the mapping is established, NBAttract often performs well when only a few new source code entities are introduced (e.g., during software evolution). However, in some cases, the F1 score is declining as the initial set becomes larger, e.g., JabRef, K9, and TeamMates (cf. Figure 1). A preliminary analysis seems to point towards overfitting, i.e., the model becomes too specific, and as a result, the recall drops (cf. Figure 3). It can also be an effect of randomness; the 1530 data points per system are pretty low considering the combinatorial complexity of random initial set sizes and compositions. However, it is sufficient to indicate the overall performance in a preliminary study such as this. The very high recall in ProM (cf. Figure 3) can be explained by the fact that the ProM framework has a very straightforward mapping, and as before, the number of data points may be too small.

We are limited to systems in Java, where the file structure often reflects the modular design of our subject systems well. While we could handle discrepancies and ambiguities well enough to create an initial set, this may not be the case in a system where the file structure is entirely different. However, we also show that these cases can use the file information. Current mapping methods, e.g., NBAttract and InMap, should likely give file information more attention.

7. Conclusions and Future Work

We found that we could construct relatively simple keywords for a majority of the 96 modules in all nine systems. Ten modules (9.6%) required weights for keywords, and 15 (15.6%) required two or more different keywords. Our mapper could successfully create an initial set using the keywords, and in some cases, this resulted in a perfect mapping.

Combining the keywords-based mapping and NBAttract using CDA provided outstanding performance with a mean precision, recall, and F1 score of 0.98, 1.0, and 0.99, respectively. The performance was higher than using random initial sets and NBAttract using CDA and file information, and the interactive technique InMap (see Table 2).

If a mapping is already established, NBAttract with CDA and file information provides good performance in many cases; however, in some systems, the model could suffer from overfitting issues (cf. Figure 3).

Using keywords is an attractive approach that can significantly reduce the mapping effort. However, a central question that remains is how to extract good candidate keywords and let a human user assign weights.

In addition, a keywords-based mapping approach is likely not applicable for some systems. We plan on performing comparative studies using the mappings from [10], where the authors claim architectural modules are not bound to the file structure of the source code.

Acknowledgments

The research was supported by the Centre for Data Intensive Sciences and Applications at Linnaeus University.

References

[1] L. De Silva, D. Balasubramaniam, Controlling software architecture erosion: A survey, Journal of Systems and Software 85 (2012) 132–151.

[2] G. C. Murphy, D. Notkin, K. Sullivan, Software reflexion models: Bridging the gap between source and high-level models, ACM SIGSOFT Software Engineering Notes 20 (1995) 18–28.

[3] N. Ali, S. Baker, R. O'Crowley, S. Herold, J. Buckley, Architecture consistency: State of the practice, challenges and requirements, Empirical Software Engineering 23 (2017) 1–35.

[4] J. Knodel, D. Popescu, A comparison of static architecture compliance checking approaches, in: The IEEE/IFIP Working Conference on Software Architecture, 2007, pp. 12–21.

[5] R. A. Bittencourt, G. Jansen de Souza Santos, D. D. S. Guerrero, G. C. Murphy, Improving automated mapping in reflexion models using information retrieval techniques, in: Working Conference on Reverse Engineering, IEEE, 2010, pp. 163–172.

[6] A. Christl, R. Koschke, M. A. Storey, Automated clustering to support the reflexion method, Information and Software Technology 49 (2007) 255–274.

[7] Z. T. Sinkala, S. Herold, Inmap: Automated interactive code-to-architecture mapping recommendations, in: IEEE 18th International Conference on Software Architecture (ICSA), 2021, pp. 173–183.

[8] T. Olsson, M. Ericsson, A. Wingkvist, Semi-automatic mapping of source code using naive bayes, in: Proceedings of the 13th European Conference on Software Architecture - Volume 2, 2019, p. 209–216.

[9] V. Tzerpos, R. C. Holt, The orphan adoption problem in architecture maintenance, in: Working Conference on Reverse Engineering, IEEE, 1997, pp. 76–82.

[10] J. Garcia, I. Krka, C. Mattmann, N. Medvidovic, Obtaining ground-truth software architectures, in: 35th International Conference on Software Engineering (ICSE), 2013, pp. 901–910.

[11] J. Buckley, N. Ali, M. English, J. Rosik, S. Herold, Real-time reflexion modelling in architecture reconciliation: A multi case study, Information and Software Technology 61 (2015) 107–123.

[12] N. Anquetil, T. C. Lethbridge, Recovering software architecture from the names of source files, Journal of Software Maintenance: Research and Practice 11 (1999) 201–221.

[13] A. Christl, R. Koschke, M. A. Storey, Equipping the reflexion method with automated clustering, in: Working Conference on Reverse Engineering, IEEE, 2005, pp. 98–108.

[14] T. Olsson, M. Ericsson, A. Wingkvist, An exploration and experiment tool suite for code to architecture mapping techniques, in: Proceedings of the 13th European Conference on Software Architecture - Volume 2, ECSA '19, 2019, p. 26–29.

[15] I. Witten, E. Frank, M. Hall, C. Pal, Data Mining, Fourth Edition: Practical Machine Learning Tools and Techniques, 4th ed., Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2016.

[16] T. Olsson, M. Ericsson, A. Wingkvist, s4rdm3x: A tool suite to explore code to architecture mapping techniques, Journal of Open Source Software 6 (2021) 2791. doi:10.21105/joss.02791.

[17] J. Brunet, R. A. Bittencourt, D. Serey, J. Figueiredo, On the evolutionary nature of architectural violations, in: Working Conference on Reverse Engineering, IEEE, 2012, pp. 257–266.

[18] J. Lenhard, M. Blom, S. Herold, Exploring the suitability of source code metrics for indicating architectural inconsistencies, Software Quality Journal (2018).

[19] A. Nurwidyantoro, T. Ho-Quang, M. R. V. Chaudron, Automated classification of class role-stereotypes via machine learning, in: Proceedings of the Evaluation and Assessment on Software Engineering, 2019, p. 79–88.

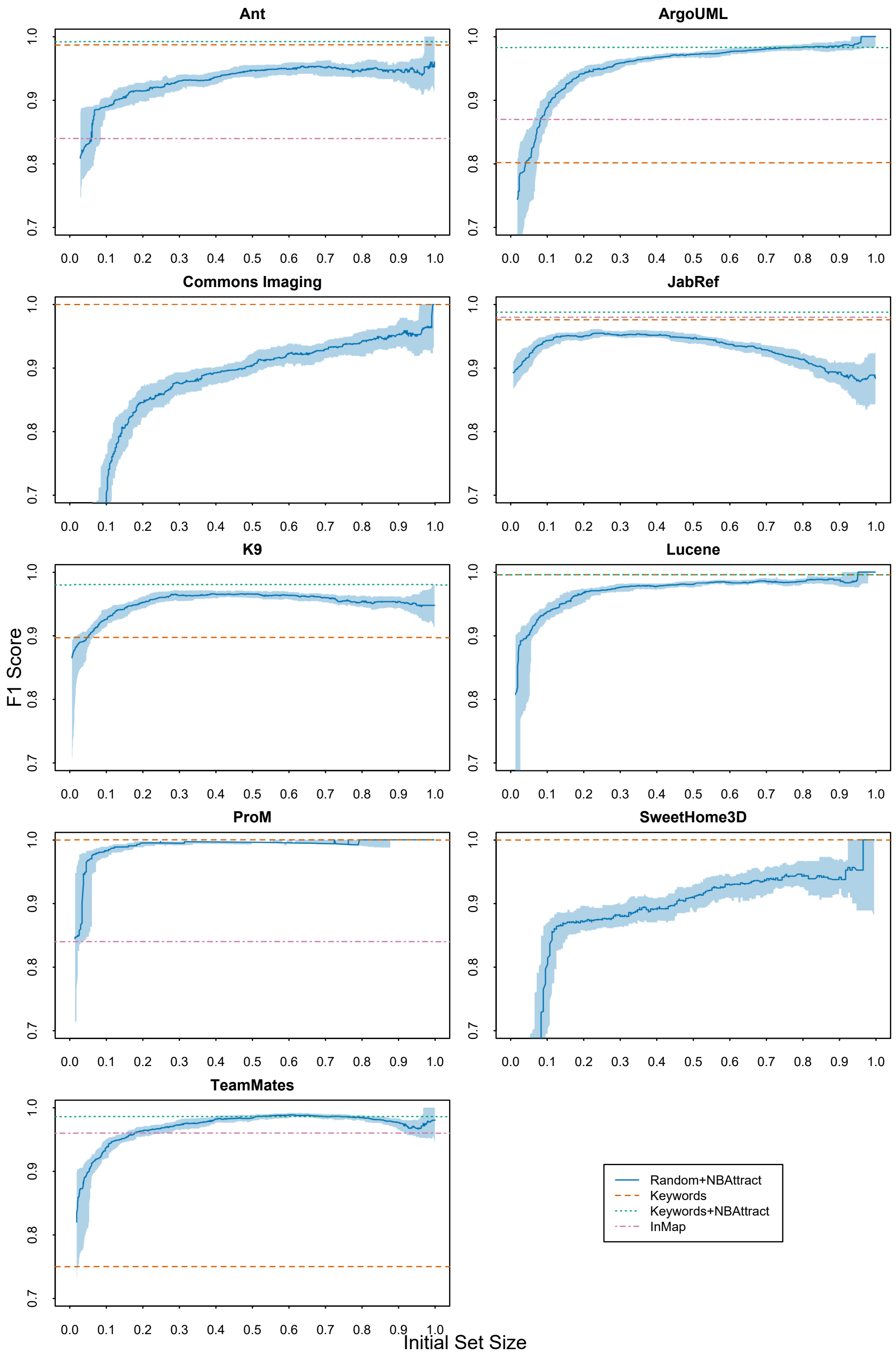

Figure 1: The F1 score of each approach, Random+NBAttract are shown with a running median and the running 25th to 75th quartiles. Note that the F1 score starts at 0.7.

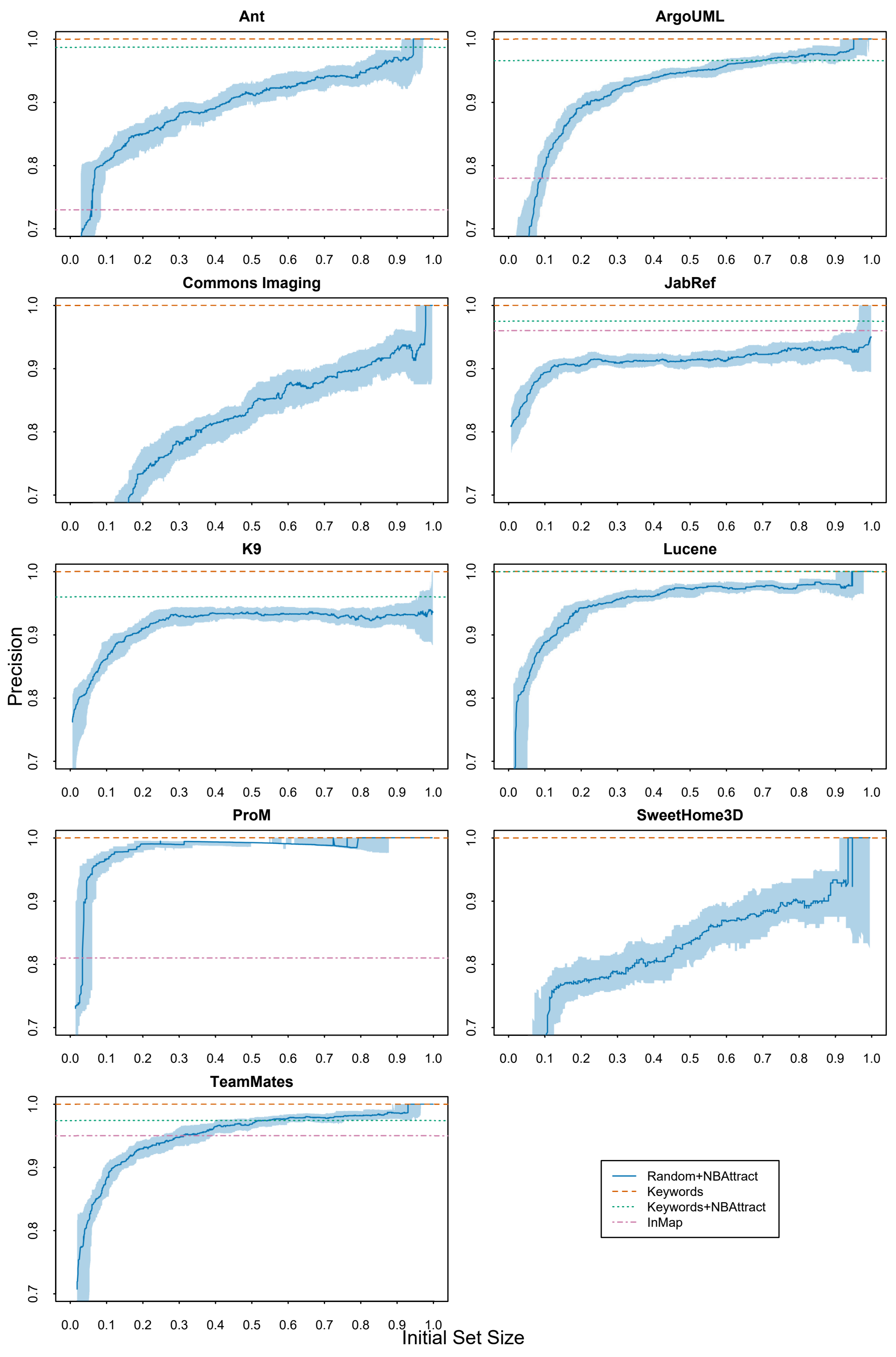

Figure 2: The precision of each approach, Random+NBAttract are shown with a running median and the running 25th to 75th quartiles. Note that the precision starts at 0.7.

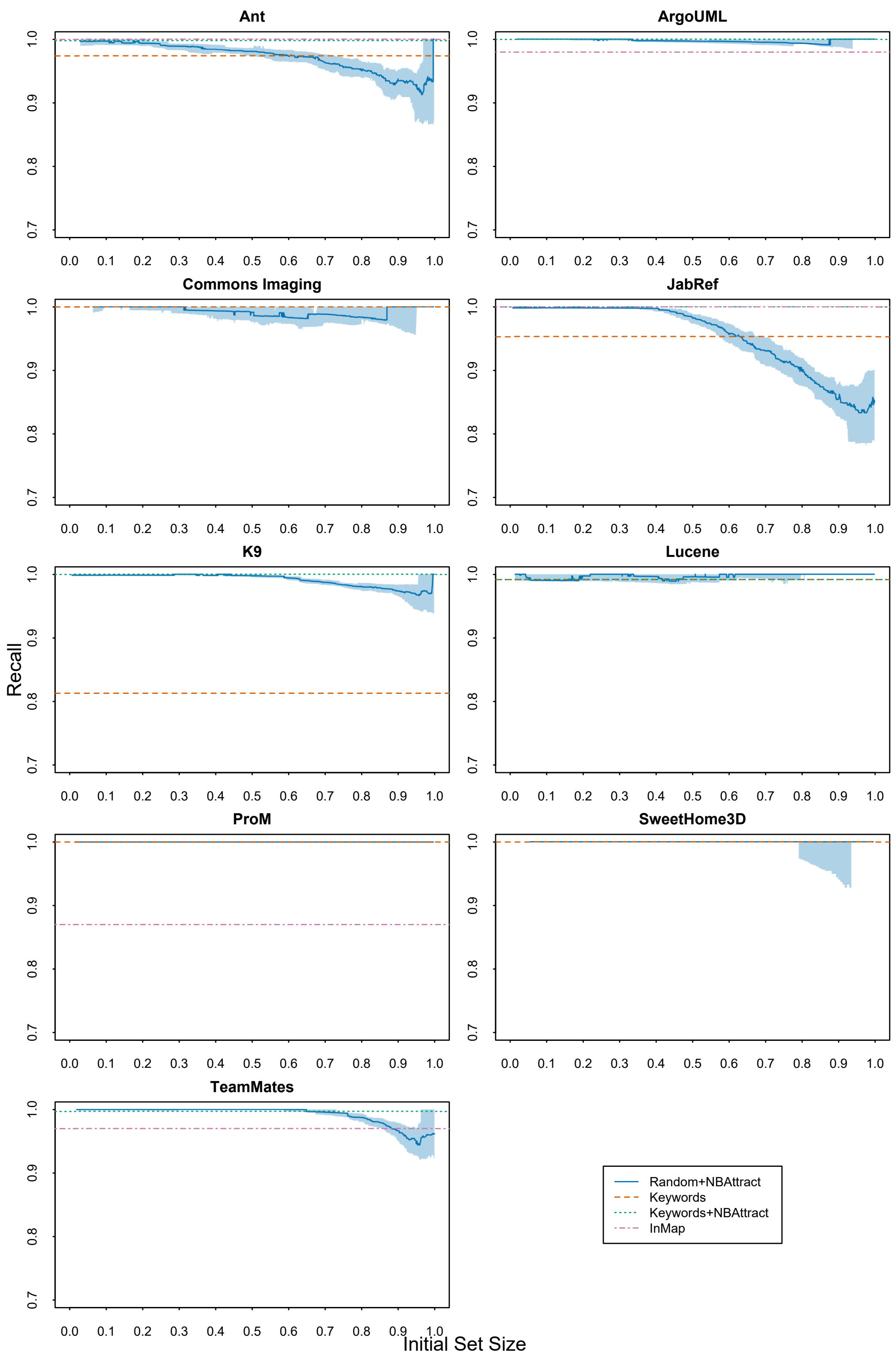

Figure 3: The recall of each approach, Random+NBAttract are shown with a running median and the running 25th to 75th quartiles. Note that the recall starts at 0.7.