Automatic examining of software architectures on mobile applications codebases

Dragoş Dobrean
Computer Science Department, *Babeş Bolyai University*, Cluj Napoca, Romania
dobrean@cs.ubbcluj.ro

***Abstract*—Mobile applications have grown to become complex software systems and some of the most used pieces of software by end users all around the world. With the increase in their complexity, software architecture has become an important topic and a pressure point in their development lifecycle. The purpose of this work is to define an automatic method for extracting and examining the software architecture of mobile applications by leveraging the use of Software Development Kits (SDKs) and Artificial Intelligence algorithms. The proposed system is used for finding architectural issues on the analysed codebase early in the development phase and provides insightful information for both software developers, architects as well as for the management team.**

***Index Terms*—mobile applications software architecture examining; automatic examination of software architectures**

I. Introduction & context

Mobile applications have grown to be complex software systems which are highly used and do heavy processing [1]. Entire companies are build around this type of software products (Whatsapp, Instagram, etc.) and they are responsible for their main source of income making these software products one of the central pieces of those businesses. In order for a mobile application to be flexible to change, offer new functionalities, implement all the new features of the SDKs, and support the high end devices (most technological advanced) as well as the low end (less technological advanced) ones those kind of applications need to have a well defined and correctly implemented software architecture.

Even if some architectural patterns make the development process less error prone with respects to architectural correctness, the architecture can not be easily imposed in a development company without an external system for validating it. There are many tools which can extract the implemented or descriptive architecture [2] from a codebase [3]–[5] but those are not tailored for mobile applications and their particularities, such as the constant usage of certain SDKs and libraries provided by the creators of mobile Operating Systems (OSs) and the fact that the most widely used architectural patterns are presentational ones derived from Model-View-Controller [6], [7].

Another limitation of some of the existing tools is that they do not provide a list of architectural issues found in the codebase or insightful metrics for mobile applications developers [8]. Furthermore they are not meant to be ran in Continuous Integration / Continuous Delivery (CI/CD) pipelines in order to provide early feedback to the developers regarding architectural inconsistencies and issues.

Our research focuses on creating a software architecture recommendation system carried out in a checker tool for mobile applications which highlights the architectural problems and provides meaningful insights regarding the codebase, such as how it has improved from the last run and the percentage of wrong dependencies per architectural layer. Moreover, it allows the specification of custom software architectures for more complex projects. In addition, the tool is meant to be run in a CI/CD pipeline as shown in Fig. 1 in order to detect issues early in the development phase, while also offering reports regarding the health of the codebase and outcomes of refactoring periods.

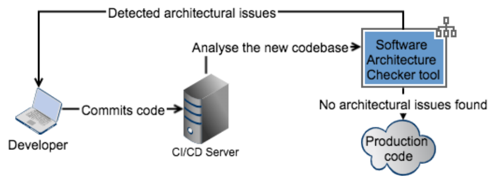
Detected architectural issues
Analyse the new codebase
Software Architecture Checker tool
Commits code
Developer
CI/CD Server
No architectural issues found
Production code

Fig. 1: Software Architecture Checker tool CI/CD workflow

The expected outcomes of this research are:

- Developing a novel approach for identifying and examining the implemented software architecture of mobile applications codebases.
- Defining a framework for specifying custom architectural patterns and evaluate mobile applications codebases against those

II. Research approach

A. Identifying and examining architectural inconsistencies in an implemented architecture

a) Motivation: In order to be able to evaluate the architectural health of a codebase, this must be split into composing components and an investigation should be conducted on the links and coupling between them. The analysis on the dependencies between the components is done using a predefined set of rules which should define the desired or prescriptive architecture [2] of the codebase.

Mobile applications are clients ergo our interest revolves around the monolithic architectures. Among the monolithic architectures we are targeting the layered one as this subsection of styles is most commonly used on mobile applications. Certain parts of the application such as the communication with the backend server or the business logic can use other micro architectures styles such as pipe-and-filter or microkernels. A first attempt is to focus on analysing Model-View-Controller (MVC) codebases as this is the precursor of most of architectural patterns used on mobile platforms such as Model-View-View-Model (MVVM) [9] or Model-View-Presenter (MVP) [10]. In this work the MVC is considered as a specialisation of the layer architectural style.

b) Research questions:

- RQ1: How can we identify the layers and their components in an MVC architecture from a mobile application codebase?
- RQ2: How can we analyse the MVC components and links between them to highlight architectural issues?
- RQ3: How accurate is the detection of MVC components?

c) Existing work: There have been many proposed approaches for extracting and reconstructing an architecture from a codebase by using lexical information [11], structural information [12], by using a mixed approach [13]–[15] or tools [16], [17]. However, none of those exclusively focuses on mobile platforms and their particularities to our knowledge.

MVC has been analysed from multiple points of view, including mobile applications by both practitioners [18]–[20] and academic community [9], [21]–[25] highlighting its strong and weak features as well as possible configurations of dependencies between layers; those findings were used for the analysis phase. There have also been numerous researches on the evaluation of the recovered architectures [9], [25], [26] and the comparison of those [27].

d) Approach: Our approach follows 3 directions, firstly we detect the components using the information provided by the SDKs, secondly we use AI algorithms, the last direction is a combination of the first two – we pursue a hybrid (combined) path.

Mobile applications are built using certain SDKs provided by the creators of mobile OSs. Those SDKs provide classes and various structures for aiding the building of UI elements and displaying them on the screen. We can use this information for deciding which elements belong to the View layer. Basically every element which inherits from an SDK defined View component (on iOS is UIView, View on Android) can be attributed to the Views layer.

In addition, we can also extract information regarding Controller layer using the same approach. On the mobile SDKs we have entities which are responsible for configuring the View elements and handling their input (UIViewController on iOS, Activity and Fragment on Android); any element which inherits from one of those should be treated as a Controller entity. The Controller layer should also contain the elements which keep the state of the application and decide when to navigate from one screen to another. Sometimes this functionality is implemented in plain objects which do not inherit from any predefined class. They are called Coordinating Controllers and can be identified by detecting all the objects which manipulate the simple Controller elements or other Coordinating Controllers. A Coordinating Controller should also be treated as Controller. After the detection the elements composing the View and Controller layers the rest of the codebase can be marked as being part of the Model layer.

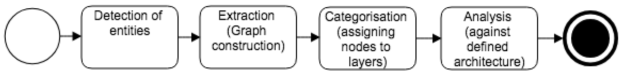
Detection of entities
Extraction (Graph construction)
Categorisation (assigning nodes to layers)
Analysis (against defined architecture)

Fig. 2: Identification & analysis workflow

This detection mechanism might not give 100% accurate results as some elements might be wrongly categorised based on the heuristics used, the order of application and the analysed architecture. The detection could be improved by using clustering algorithms such as K means [28] or Mean Shift [29]. All the detected codebase elements should be clustered into groups based on their dependencies. This method will categorise the elements of the codebase based on their dependencies providing information regarding their actual behaviour in contrast to the declared behaviour extracted using the SDK inheritance method.

By using the SDK inheritance detection method we categorise the elements based on their declared behaviour and we can extract with a high degree of confidence the elements from the View and Controller layer. For the other layers the clustering comes into play and helps distributing the codebase based on their dependencies. Such hybrid approach allows the proposed system to analyse simpler architectures such as MVC but also its flavours (MVVM, MVP) and more complex architectures such as View-Interactor-Presenter-Router-Entity (VIPER) [30].

For the analysis part we construct the topological structure of the architecture (an oriented graph) in which every element (an element is a piece of a software systems: a class, struct, enum, protocol/interface or a module [31]) from each layer represents a node, the links between nodes represent dependencies between those elements found in the codebase. Based on the MVC flavour used, rules can be extracted regarding the allowed dependencies between the layers, afterwards the graph is checked for links which invalidate those rules, the complete process is shown in Fig. 2.

e) Evaluation: In order to validate the proposed approach, the experiments should be conducted on multiple different sized codebases which are using distinct architectures on both open-source and private source projects in two phases:

1) Detect and analyse the codebases using only the SDK inheritance detection method
2) Add the clustering process to the previous method and re-conduct the experiments

Before conducting the experiments, the architecture of the selected codebases should be manually extracted and used as ground truth for the detection process.

B. Constraint based system for defining a personalised architecture

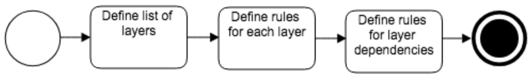
Define list of layers
Define rules for each layer
Define rules for layer dependencies

Fig. 3: Constraints system workflow

a) Motivation: Certain projects have particularities which do not allow the usage of common software architectures. One example could be projects which use certain libraries of UI components rather than using the mobile SDKs provided ones, projects which have custom architectural layers or projects which use a more specialised architecture such as VIPER. Detecting the used architecture from those codebases is a difficult task which can not be automatically achieved as there are many particularities and an exhaustive list of mobile applications architectural patterns can not be constructed. By using a constraint based system for specifying the personalised architecture we can also incorporate the knowledge of the architecture [32] and further use it for the architectural evolution as a future direction of study.

b) Research questions:

- RQ1: What are the elements required for defining a software architectural pattern in the scope of the software architecture checker system?
- RQ2: Can a more accurate specification improve the accuracy of the checker system?

c) Existing work: There are many ways of specifying an architecture using specialised languages [33]–[35] and previous researches have also been done for specifying architectural constraints [36]. However those approaches are too complex from a practitioner point of view. Researches have also been done on dynamic ADLs used for endorsing the architectural evolution [34]. There is a trade-off between the expressiveness of a general-purpose ADL and the optimisation and personalisation of a more specialised ADL. Moreover, those approaches refer to distributed or enterprise software systems which are a few orders of magnitude more complex than a mobile application. Furthermore, to the best of our knowledge, these do not have any sort of information regarding the matching of certain codebase entity within a certain architectural layer which is a must have feature in respects to our research goal.

d) Approach: In order to be able to analyse custom defined architectures we need to have a system for specifying the architectural rules as well as how the codebase elements should be distributed in layers. The proposed approach (Fig. 3) is aimed to define a list of layers of the architecture together with rules needed for assigning elements from the codebase to those. The rules specify different properties of the elements, whether or not they should inherit from a certain type, their file path in the codebase, as well as types of properties and method definitions used. By creating layers and specifying the matching rules for the elements composing those, the codebase can be analysed with a higher degree of confidence. This approach offers the flexibility to specify any architectural pattern while also detecting with precision which elements should reside in which layer. In order to fully specify the software architecture we would need to define the consistent dependencies between layers as directional links between them. Those matching rules and dependencies are also aimed to guide and support the evolution of architectures.

e) Evaluation: For the validation of the constraint based system the proposed approach will be used in multiple open-source and private projects which use unpopular or custom architectures. The applications analysed in the previous phase of the study will be re-analysed in order to see how the accuracy of the system is improved. Experiments will be conducted using different sets of rules in order to see which ones impact the accuracy of the checker system the most.

III. Timeline

The presented research goal is aligned with my PhD proposal. The detection and analysis part using SDK inference was already implemented and the first results indicate an average accuracy of 89.6% of the evaluation process on several mobile codebases of different sizes. The complete results, a detailed description of the workflow and experiments conducted have been published [37], [38]. The current focus of the research is on the clustering, the proposed method seems promising and papers with our findings will be published later this year. The last part of the study will be tackled as soon as the insights about the clustering part are obtained.

The tool built for the validation of the proposed approach can be easily be integrated into a CI/CD pipeline as it static analyses the codebase. After insights are obtained from the last part of the study, we plan to validate it with software companies on ongoing projects.

IV. Future directions

Our plans for the future revolve around the evolution of mobile architectures. It is common in the development of the mobile applications for projects to have an architectural pattern in place, but which needs to be changed due to technology considerations (supporting certain libraries or features), business (short development cycles or the expansion of the team), or architectural ones (architectural erosion and architectural drift).

In the last few years extensive work has been done in the domain of evolutive software architectures [39]–[43] and Artificial Intelligence (AI) software architecture improvements [44] as well as in creating tools to aid the process of evolution [45], [46]. Those researches try to shed some light on how a software architecture can evolve and how smart algorithms can be used for improving the architectural health of codebases. If the results from the proposed approach are promising we plans to further that work and apply its value in the domain of mobile applications by using the already existing models and enhancing them with mobile development particularities such as the constant usage of SDKs and the predominantly usage of presentational architectures.

We are especially interested in the following questions and we plan to tackle them as soon as the scope of the current research is finished

1) How can a codebase be evolved in a certain architectural pattern?
2) How can the architectural evolution of a codebase be monitored?

Acknowledgment

I thank my PhD advisor, Professor PhD Laura Dioşan from Babeş Bolyai University, Cluj Napoca, Romania, for all the valuable academic insights, support and motivation offered while pursuing the research goal presented in this paper.

References

[1] M. Zhang. (2018) This is the full photoshop coming to the ipad. [Online]. Available: https://petapixel.com/2018/10/15/this-is-the-full-photoshop-coming-to-the-ipad/

[2] J. Garcia, D. Popescu, G. Edwards, and N. Medvidovic, "Identifying architectural bad smells," in *Software Maintenance and Reengineering, 2009. CSMR'09. 13th European Conf. on*. IEEE, 2009, pp. 255–258.

[3] N. Anquetil and T. C. Lethbridge, "Recovering software architecture from the names of source files," *Journal of Software Maintenance: Research and Practice*, vol. 11, no. 3, pp. 201–221, 1999.

[4] G. Antoniol, R. Fiutem, and L. Cristoforetti, "Design pattern recovery in object-oriented software," in *Program Comprehension, 1998. IWPC'98. Proc., 6th Int. Workshop on*. IEEE, 1998, pp. 153–160.

[5] L. Ding and N. Medvidovic, "Focus: A light-weight, incremental approach to software architecture recovery and evolution," in *Software Architecture, 2001. Proc. Working IEEE/IFIP Conf. on*. IEEE, 2001, pp. 191–200.

[6] M. Fowler, *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[7] T. Reenskaug, "The model-view-controller (mvc) its past and present," *University of Oslo Draft*, 2003.

[8] D. M. Le, P. Behnamghader, J. Garcia, D. Link, A. Shahbazian, and N. Medvidovic, "An empirical study of architectural change in open-source software systems," in *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on*. IEEE, 2015, pp. 235–245.

[9] R. Garofalo, *Building enterprise applications with Windows Presentation Foundation and the Model View View Model Pattern*. Microsoft Press, 2011.

[10] M. Potel, "Mvp: Model-view-presenter the taligent programming model for c++ and java," *Taligent Inc*, p. 20, 1996.

[11] A. Corazza, S. Di Martino, V. Maggio, and G. Scanniello, "Weighing lexical information for software clustering in the context of architecture recovery," *Empirical Software Engineering*, vol. 21, no. 1, pp. 72–103, 2016.

[12] M. Pinzger and H. Gall, "Pattern-supported architecture recovery," in *Proceedings 10th International Workshop on Program Comprehension*. IEEE, 2002, pp. 53–61.

[13] J. Garcia, I. Ivkovic, and N. Medvidovic, "A comparative analysis of software architecture recovery techniques," in *Proc. of the 28th IEEE/ACM Int. Conf. on Automated Software Engineering*. IEEE Press, 2013, pp. 486–496.

[14] A. Boaye Belle, "Recovering software layers from object oriented systems: a formalization as an optimization problem," Ph.D. dissertation, École de technologie supérieure, 2016.

[15] S. M. Naim, K. Damevski, and M. S. Hossain, "Reconstructing and evolving software architectures using a coordinated clustering framework," *Automated Software Engineering*, vol. 24, no. 3, pp. 543–572, 2017.

[16] F. A. Fontana and M. Zanoni, "A tool for design pattern detection and software architecture reconstruction," *Information sciences*, vol. 181, no. 7, pp. 1306–1324, 2011.

[17] N. Ghorbani, J. Garcia, and S. Malek, "Detection and repair of architectural inconsistencies in java."

[18] B. Orlov. (2015) ios architecture patterns: Demystifying mvc, mvp, mvvm and viper. [Online]. Available: https://medium.com/ios-os-x-development/ios-architecture-patterns-ecba4c38de52

[19] K. Kocsis. (2018) Architectural patterns, mvc, mvvm: What is the hype all about? [Online]. Available: http://kristofk.com/mvc-mvvm-viper/

[20] D. DeLong. (2017) A better mvc. [Online]. Available: https://davedelong.com/blog/2017/11/06/a-better-mvc-part-1-the-problems/

[21] A. Daoudi, G. ElBoussaidi, N. Moha, and S. Kpodjedo, "An exploratory study of mvc-based architectural patterns in android apps," in *Proc. of the 34th ACM/SIGAPP Symposium on Applied Computing*. ACM, 2019, pp. 1711–1720.

[22] D. Plakalovic and D. Simic, "Applying mvc and pac patterns in mobile applications," *arXiv preprint arXiv:1001.3489*, 2010.

[23] H. J. La and S. D. Kim, "Balanced mvc architecture for developing service-based mobile applications," in *e-Business Engineering (ICEBE), 2010 IEEE 7th Int. Conf. on*. IEEE, 2010, pp. 292–299.

[24] F. E. Shahbudin and F.-F. Chua, "Design patterns for developing high efficiency mobile application," *Journal of Information Technology & Software Engineering*, vol. 3, no. 3, p. 1, 2013.

[25] T. Olsson, M. Ericsson, and A. Wingkvist, "Towards improved initial mapping in semi automatic clustering," in *Proc. of the 12th European Conf. on Software Architecture: Companion Proc.* ACM, 2018, p. 51.

[26] M. Abboud, H. Naja, M. Oussalah, and M. Dbouk, "Kdd extension tool for software architecture extraction," in *15th IEEE Int. Conf. on Software Engineering Research and Practice*, 2017.

[27] T. Lutellier, D. Chollak, J. Garcia, L. Tan, D. Rayside, N. Medvidović, and R. Kroeger, "Measuring the impact of code dependencies on software architecture recovery techniques," *IEEE Transactions on Software Engineering*, vol. 44, no. 2, pp. 159–181, 2018.

[28] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[29] Y. Cheng, "Mean shift, mode seeking, and clustering," *IEEE transactions on pattern analysis and machine intelligence*, vol. 17, no. 8, pp. 790–799, 1995.

[30] MutualMobile. (2014) Meet viper: Mutual mobile?s application of clean architecture for ios apps. [Online]. Available: https://mutualmobile.com/posts/meet-viper-fast-agile-non-lethal-ios-architecture-framework

[31] J. M. Barnes, D. Garlan, and B. Schmerl, "Evolution styles: foundations and models for software architecture evolution," *Software & Systems Modeling*, vol. 13, no. 2, pp. 649–678, 2014.

[32] O. Le Goaer, D. Tamzalit, M. C. Oussalah, and A.-D. Seriai, "Evolution styles to the rescue of architectural evolution knowledge," in *Proc. of the 3rd international workshop on Sharing and reusing architectural knowledge*. ACM, 2008, pp. 31–36.

[33] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified modeling language reference manual, the*. Pearson Higher Education, 2004.

[34] F. Oquendo, "π-adl: an architecture description language based on the higher-order typed π-calculus for specifying dynamic and mobile software architectures," *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 3, pp. 1–14, 2004.

[35] S. Hussain, "Investigating architecture description languages (adls) a systematic literature review," 2013.

[36] S. Kallel, C. Tibermacine, S. Kallel, A. H. Kacem, and C. Dony, "Specification and automatic checking of architecture constraints on object oriented programs," *Information and Software Technology*, vol. 101, pp. 16–31, 2018.

[37] D. Dobrean and L. Dioşan, "Model view controller in ios mobile applications development," in *Proc. of the 31st International Conference on Software Engineering Knowledge Engineering*, 2019, p. accepted.

[38] ——, "An analysis system for mobile applications mvc software architectures," in *Proc. of the 14th International Conference on Software Technologies*, 2019, p. accepted.

[39] N. Ford, R. Parsons, and P. Kua, *Building Evolutionary Architectures: Support Constant Change*. " O'Reilly Media, Inc.", 2017.

[40] D. Tamzalit and T. Mens, "Evolution patterns: Designing and reusing architectural evolution knowledge to introduce architectural styles," *arXiv preprint arXiv:1605.06289*, 2016.

[41] C. E. Cuesta, E. Navarro, D. E. Perry, and C. Roda, "Evolution styles: using architectural knowledge as an evolution driver," *Journal of Software: Evolution and Process*, vol. 25, no. 9, pp. 957–980, 2013.

[42] B. Graaf, *Model-driven evolution of software architectures*. IEEE, 2007.

[43] A. Hassan and M. Oussalah, "Evolution styles: Multi-view/multi-level model for software architecture evolution," *Journal of Software*, vol. 13, no. 3, pp. 146–155, 2018.

[44] A. Martens, H. Koziolek, S. Becker, and R. Reussner, "Automatically improve software architecture models for performance, reliability, and cost

using evolutionary algorithms," in *Proc. of the first joint WOSP/SIPEW international conference on Performance engineering*. ACM, 2010, pp. 105–116.

[45] S. Getir, L. Grunske, A. van Hoorn, T. Kehrer, Y. Noller, and M. Tichy, "Supporting semi-automatic co-evolution of architecture and fault tree models," *Journal of Systems and Software*, vol. 142, pp. 115–135, 2018.

[46] D. Garlan and B. Schmerl, "Ævol: A tool for defining and planning architecture evolution," in *Proc. of the 31st Int. Conf. on Software Engineering*. IEEE CS, 2009, pp. 591–594.