Intelligent methods for inferring software architectures from mobile applications codebases

Ph.D. Student: **Dragoş Dobrean**,
Supervisor: Professor Ph.D. **Laura Dioşan**

November 19, 2021

Computer Science Department, **Babes Bolyai University**, Cluj Napoca, Romania
MECO research group - Applied Computational Intelligence Research Institute

Table of contents

Introduction & Context

Introduction

- Over **two-thirds** of the world's population are using smartphone devices
- Smartphones have become our most personal device
- Average users spend around **4.2 hours** each day in mobile applications [1]

[1] Sarah Perez, **Consumers now average 4.2 hours per day in apps, up 30% from 2019**, TechCrunch, April 8, 2021

Introduction

- A lot of companies were built around mobile applications (WhatsApp, Instagram, Tinder, Snapchat)
- Mobile applications are one of the most commonly written pieces of software nowadays
- As the technology advances, mobile applications become more complex (audio/photo/video processing, Augmented Reality, Machine Learning, databases)

What is software architecture?

"Architecture is concerned with the selection of architectural elements, their interactions, and the constraints on those elements and their interactions necessary to provide a framework in which to satisfy the requirements and serve as a basis for the design." [2]

[2] Dewayne E Perry and Alexander L Wolf. **Foundations for the study of software architecture.** ACM SIGSOFT Software engineering notes, 17(4):4052, 1992.

Context

- Mobile applications usually start as Minimum Viable Projects projects
- If they evolve into more complex products, **most of the time don't have an architectural pattern in place**
- They are developed by different people (different backgrounds, skills)

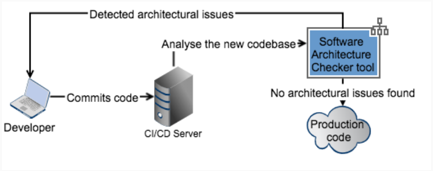
Software architecture in mobile projects

- **100%** of developers and **80%** of students encounter architectural issues, **70%** of developers on a monthly basis [3]
- The right architecture for the application based on the functional requirements and the roadmap of the application can have a strong impact on the overall **cost** of the project
- Enables **extensibility**, and **flexibility** to adapt to new technologies, and a higher number of devices on which the application can run on
- **Security** and **scalability** of the application as well as the ability to easily provide **new and interactive user interfaces and experiences**

[3] Dobrean, D., Dioşan, L. **Importance of software architectures in mobile projects**, SACI, May 19-21 2021, to be published in IEEE

Goals

- Our **end-goal** is to build a software architecture checker system for mobile codebases
- Automatically inferring the software architectures from mobile codebases is one of the cornerstones of the checker system
- Using the information from Software Development Kits (SDKs) and Machine Learning techniques


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

Mobile architecture checker system in a CI/CD pipeline [4]

[4] Dobrean, D., **Automatic Examining of Software Architectures on Mobile Applications Codebases**, (IEEE International Conference on Software Maintenance and Evolution (pp. 595-599)) *(ISI, categ. A, **8** points, IEEE indexed)*

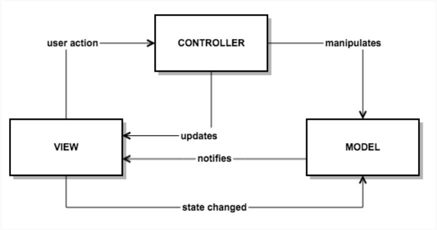
Model View Controller

Description

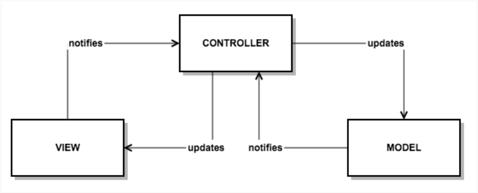
- Mobile applications run on the client-side, ergo, they should use presentational architectural patterns which **generally descend MVC**
- MVC is one of the **most commonly used** architectures for developing mobile applications [5]
- **Model** - all the business logic, data access, and mapping of the data
- **View** - displays the data in different forms based on the scope of the application and its requirements
- **Controller** - input logic and acting as a proxy between the View and Model layer

[5]Chris Hefferman , Dragos Dobrean, Dave Vewer, Benjamin Hendricks, **iOS Developer Survey 2019-2021**

Classic Model View Controller



Classic Model View Controller architectural overview



Apple’s Model-View-Controller architectural overview

Common issues

- **Complexity** – the components might become bloated with code, if not split correctly (massive view controllers) [6]
- **Misunderstandings** – different flavors of MVC from various platforms (Web, Dekstop) with different allowed dependencies [6]
- **Design issues** – design pattern issues, the violation of SOLID principles, Co-change Coupling, Smells, etc. [6]

[6]Dobrean, D., Dioşan, L., **Model View Controller in iOS mobile applications development**, Proc. of the 31st International Conference on Software Engineering Knowledge Engineering, 2019, 547-552. *(ISI, categ. B, **4** points, SCOPUS indexed)*

Scientific problem

Definition

In this work, we focus on finding a way of automatically inferring architectural layers from mobile codebases.

- $A = \{a_1, a_2, \ldots, a_n\}$ where a_i, $i \in \{1, 2, \ldots, n\}$ denotes a component
- The purpose of all our proposals is to find a way for splitting the codebase into architectural layers
- Architectural layer – a partition P of the components $P = \{P_1, P_2, \ldots, P_m\}$ in A
- In the case of MVC, $P = \{P_1, P_2, P_3\}$, P_1 represents the Model layer, P_2 the View layer, and P_3 the Controller layer.
- Each component a_i ($i \in \{1, 2, \ldots, n\}$) could be represented by one or more features or characteristics – $F(a_i) = [F_i^1, F_i^2, \ldots, F_i^k]$.

Challenges

1. **distinct domains**
2. **fast-paced environment**
3. **developer mistakes**
4. **external libraries**
5. **legacy code**
6. **different sized applications/codebases**
7. **team changes**
8. **different variations of MVC**
9. **no set of iOS benchmark applications**

Proposed approaches

Deterministic approach

mobile ArchCheckSys (mACS) [7]

- Uses information from the **mobile SDKs** and the **codebase** for inferring architectural layers
- **Simple formalization** of the architectural rules
- **Automated** workflow

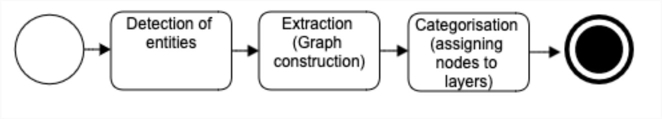
[7] Dobrean, D., Dioşan, L., **An analysis system for mobile applications MVC software architectures**, (Proceedings of the 14th International Conference on Software Technologies, pages 178-185) *(ISI, categ. B,* ***4*** *points, Web Of Science, SCOPUS indexed)*

mobile ArchCheckSys (mACS)

Challenges

- Architecture detection by using only the information obtained from the mobile SDKs
- A method able of identifying layers of software components that can not be inferred from the SDKs

mobile ArchCheckSys (*mACS*)



Detection of entities
Extraction (Graph construction)
Categorisation (assigning nodes to layers)

mACS workflow

- Analysis **MVC** architectures
- Leverages **SDK information** for architecture extraction
- **Automated** process
- Language and platform **agnostic**

mACS – Detection

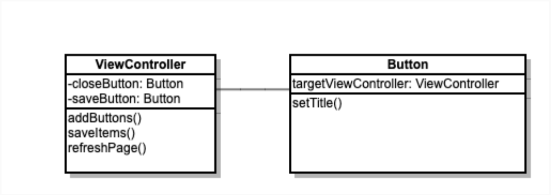
- **Static** analysis
- Components
    - **Classes**
    - **Structs**
    - **Protocols**
    - **Enums**
- Public and private **properties**
- Public and private class and instance **methods**
- The output of this phase is stored in a **structured** file (JSON/XML)
- The Abstract Syntax Tree (**AST**) can be used for generating the file

mACS – Extraction

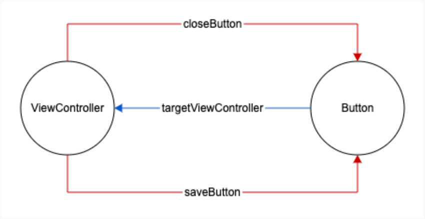
It creates a **dependency graph** based on the information in that file, each node contains the following information:

- Name
- Type
- Inherited type
- Instance and class variables (name and type)
- Instance and class methods (together with parameters names and types)
- Path

mACS – Extraction - example



Example of a UML diagram for codebase components.



A directed graph showcasing the edges for the example UML diagram.

mACS – Categorisation

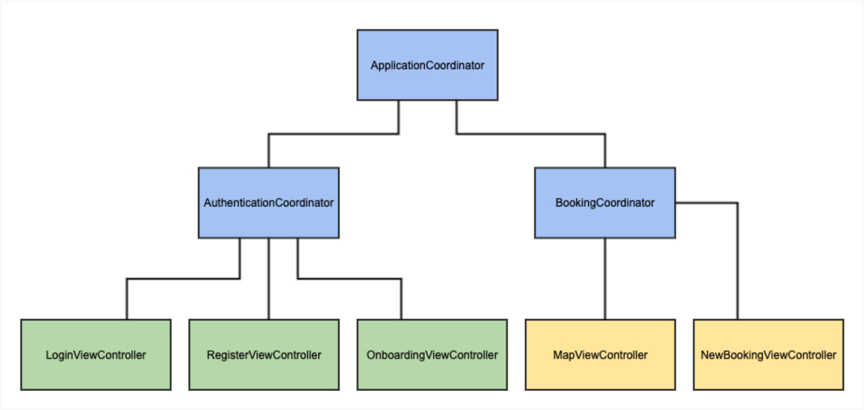
Heuristics

$$\begin{aligned} \mathit{Controller} =& \{n \mid \mathit{pred}_X(n, \mathit{SDK's Controller}) = \mathit{True}, \\ & \text{where } X \in \{\mathit{instanceOf}, \mathit{inheritance}\}\} \end{aligned} \tag{1}$$

$$\begin{aligned} \mathit{View} =& \{n \mid \mathit{pred}_X(n, \mathit{SDK's View}) = \mathit{True}, \\ & \text{where } X \in \{\mathit{instanceOf}, \mathit{inheritance}\}\} \end{aligned} \tag{2}$$

$$\mathit{Model} = \mathit{Components} - \mathit{View} - \mathit{Controller} \tag{3}$$

mACS – Categorisation - example



Coordinating controllers flows

mACS – Categorisation

Coordinating controllers

$$\begin{aligned}
Coordinators = \{n | \exists v \in n.properties \text{ and } c \in Controller \\
\text{such as } pred_{instanceOf}(v, c) = True \text{ or } \exists m \in n.methods \\
\text{and } c \in Controller \text{ such as } pred_{using}(m, c) = True\}
\end{aligned} \tag{4}$$

$$Controller = Controller \cup Coordinators \tag{5}$$

Non-deterministic approach

Clustering ARchitecture Layers (*CARL*) [8]

- A **novel approach** for automatically detecting architectural layers
- Unsupervised **Machine Learning** method
- Leverages information from both the **codebase** and the **SDKs**

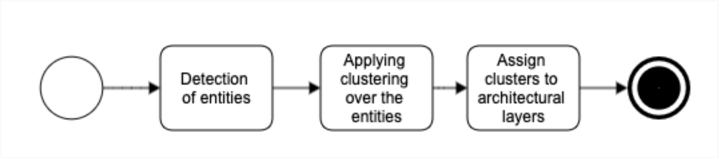
[8]Dobrean, D., Dioşan, L. **Detecting Model View Controller Architectural Layers using Clustering in Mobile Codebases**, (Proceedings of the 15th International Conference on Software Technologies (2020), pages 196-203) *(ISI, categ. B,* ***4*** *points, Web Of Science, SCOPUS indexed)*

Clustering ARchitecture Layers (*CARL*)

Challenges

- Architecture detection by using Machine Learning algorithms
- Clustering method for identifying layers of software components and quick processing
- Assigning semantics to the identified clusters

Clustering ARchitecture Layers (*CARL*)


Detection of entities
Applying clustering over the entities
Assign clusters to architectural layers

CARL system phases

- **Unsupervised** method for detecting architectural layers
- **Autonomous** – no developer involvement needed
- Paves the way for custom architectures support
- Uses **hierarchical** algorithms for clustering
- The **detection** step is identical to the one from *mACS*

CARL – Pre-processing and feature extraction

The first phase from the clusterization process

Component features

- name of the component
- type (class, struct, protocol, extension)
- inherited types
- path of the component
- all public/private static and non-static methods and properties

Other features

- SDK elements involvement
- number of dependencies between the components
- presence of dependencies between components
- name similarities

CARL – Clustering

Mechanism

- connectivity bases clustering (**Agglomerative Clustering**)
- hierarchical, bottom-up
- **Euclidean distance**

Output

- **3 clusters** (Model, View, Controller)

CARL – Assigning responsibilities to the layers

- focuses on **MVC**
- leverages information from the **SDK**
- needs to be revisited in case of custom architectures (**architecture-specific heuristics**)

Hybrid approach

Hybrid Detection (*HyDe*) [9]

- **Unsupervised** – there is no prior knowledge needed before analyzing a codebase
- **Autonomous** – no developer involvement needed after the process is started
- **Two-step process** – the deterministic step (*mACS*) and non-deterministic step (*CARL*)

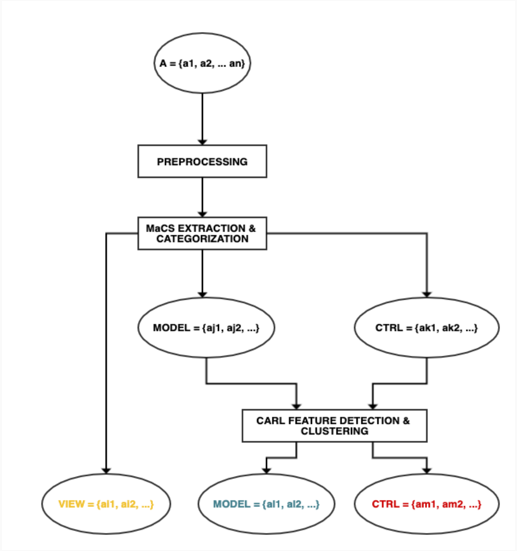
[9]Dobrean, D., Dioşan, L. **A hybrid approach to MVC architectural layers analysis**, (Proceedings of 16th International Conference on Evaluation of Novel Approaches to Software Engineering (2021)) *(ISI, categ. B,* ***4*** *points, Web Of Science, SCOPUS indexed)*, earned **Best Student Paper Award**

Hybrid Detection (*HyDe*)

Challenges

- Architecture detection using a combined approach (deterministic + non-deterministic)
- A clustering process that works while combined with the information obtained from the deterministic process

Hybrid Detection (*HyDe*)



Overview of the *HyDe* workflow.

HyDe – Deterministic step (*mACS*)

Detection & Extraction

- The topological structure of the codebase is created
- Nodes – components
- Links between nodes – dependencies
- This step is identical to the one from *mACS*

Categorization

- The codebase is distributed into 3 architectural layers (Model, View, Controller)
- Uses the information obtained from the SDK
- A set of heuristics is applied

HyDe – Non-deterministic step (*CARL*)

***HyDe* is applying the clustering process only to the Model and Controller layer obtained from the deterministic approach.**

Feature extraction

- Inclusion in the Controller layer
- Usage other Controller elements
- Distance between components names
- Number of common properties and methods

Clusterization

- Agglomerative Clustering
- 2 output clusters (Model, Controller)

Approaches overview

Approach	Information			Analysis			Granularity			Language dependent	Platform
	Structural	Lexical	SDK	Static	Heuristics	Clustering	Package	Module	Class		
belle2013layered	✓	✓	-	-	✓	-	✓	-	-	No	not mobile
belle2016combining	✓	✓	-	-	✓	-	✓	-	-	No	not mobile
belle2014recovering	✓	✓	-	-	✓	-	✓	-	-	No	not mobile
el2012reconstructing	✓	-	-	-	✓	-	✓	-	-	No	not mobile
laval2013ozone	✓	-	-	-	✓	-	✓	-	-	Yes (Java, C#, C++ and Smalltalk)	not mobile
muller1993reverse	✓	-	-	✓	-	-	-	✓	-	Yes (C, COBOL)	not mobile
rathee2017software	✓	✓	-	-	-	✓	-	-	✓	Yes (Java)	not mobile
sangal2005using	✓	-	-	✓	-	-	-	-	✓	Yes (Java)	not mobile
sarkar2009discovery	✓	-	-	-	-	✓	-	✓	-	Yes (C/C++)	not mobile
scanniello2010using	✓	✓	-	-	-	✓	-	✓	-	Yes (Java)	not mobile
schmidt2012auto	✓	-	-	-	✓	-	-	✓	-	Yes (Java)	not mobile
zapalowski2014rev	✓	-	-	-	✓	-	-	-	✓	Yes (Java)	not mobile
campos2015unve	✓	✓	-	✓	-	-	-	-	✓	Yes (Java)	Android
daoudi2019explo	✓	✓	-	-	✓	-	-	-	✓	Yes (Java/Kotlin)	Android
mACS	✓	-	✓	-	✓	-	-	-	✓	No	platform agnostic
CARL	✓	✓	✓	-	-	✓	-	-	✓	No	platform agnostic
HyDe	✓	✓	✓	-	✓	✓	-	-	✓	No	platform agnostic

Comparison of the detection mechanisms with other approaches. [10]

[10] Numerical comparison is not possible as most of the approaches are not mobile-related, or they don't give implementation / testing details

Analysis

Analysis – Data

Application	Blank	Comment	Code	#comp	Class
Demo	785	424	3364	27	Small
Game	839	331	2113	37	Small
Stock	1539	751	5502	96	Medium
Education	1868	922	4764	105	Medium
Wikipedia	6933	1473	35640	253	Medium
Trust	4772	3809	23919	403	Large
E-Commerce	7861	3169	20525	433	Large
Firefox	23392	18648	100111	514	Large

Codebases split by number of components

Analysis – Metrics

Supervised Machine Learning

- Precision
- Recall
- Accuracy
- Homogeneity
- Completeness

Unsupervised Machine Learning

- Adjusted Rand Index
- Mean Silhouette Coefficient
- Davies Bouldin Index

Analysis – Supervised Machine Learning metrics

Approach	Size	Model %		View %		Controller %		Average %		Accuracy %	Homog.	Compl.
		Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall			
SimpleCateg.	small	100	100	100	100	100	100	100	100	100	100	100
CoordCateg.	small	100	100	100	100	100	100	100	100	100	100	100
CARL	small	87	95	75	100	100	75	87	90	93	77	84
HyDe	small	93	89	100	100	63	80	85	89	88	65	61
SimpleCateg.	medium	64	100	100	72	99	57	88	76	76	47	61
CoordCateg.	medium	80	88	100	72	88	98	89	86	89	51	56
CARL	medium	66	93	83	39	93	68	81	67	74	35	45
HyDe	medium	82	82	100	72	81	93	88	82	84	55	59
SimpleCateg.	large	83	99	100	88	99	54	94	80	87	66	76
CoordCateg.	large	92	81	100	88	63	87	85	85	85	64	61
CARL	large	82	88	79	88	78	59	80	78	81	48	51
HyDe	large	94	82	100	86	65	88	86	85	86	60	57

Average (on complexity classes) precision, recall, accuracy, Homogeneity, and Completeness of the analyzed codebases against the ground truth for all detection methods: *mACS*, *CARL* and *HyDe*.
Note: *SimpleCateg.* & *CoordCateg.* refer to the two *mACS* variations – without coordinators controllers detections respectively with coordinating controllers.

Analysis – Unsupervised Machine Learning metrics

Approach	**Size**	**Adjusted Rand Index**	**Mean Silhouette coefficient**	**Davies Bouldin index**
CARL	small	80	92	18
HyDe	small	61	69	87
CARL	medium	33	78	37
HyDe	medium	57	76	36
CARL	large	50	76	40
HyDe	large	57	65	57

Adjusted Rand Index, Mean Silhouette Coefficient, and Davies Bouldin Index for the approaches which use clustering algorithms (*CARL*, *HyDe*).

Portability – Android and new iOS SDK

Codebase	Approach	Model %		View %		Controller %		Accuracy %
		Precision	Recall	Precision	Recall	Precision	Recall	
iOS - old SDK	SimpleCateg.	50	100	100	100	100	40	70
	CoordCateg.	75	100	100	100	100	80	90
iOS - new SDK	SimpleCateg.	50	100	100	100	100	40	70
	CoordCateg.	100	100	100	100	100	100	100
Android	SimpleCateg.	100	100	100	100	100	100	100
	CoordCateg.	100	100	100	100	100	100	100

The effectiveness of the categorization process in terms of Accuracy, Precision, and Recall for the iOS/Android demo apps.

Portability – macOS

Codebase	Approach	Model %		View %		Controller %		Accuracy %
		Precision	Recall	Precision	Recall	Precision	Recall	
Industrial	SimpleCateg.	62	100	100	100	100	39	78
Tracking	CoordCateg.	96	100	100	95	96	96	96
Sustainable	SimpleCateg.	58	100	100	100	100	68	82
Lifestyle	CoordCateg.	91	100	100	100	100	96	97

The effectiveness of the categorization process in terms of Accuracy, Precision, and Recall for the macOS apps.

Conclusions & Further work

SWOT analysis

- **Strengths**
    - Good performance, use the information from SDKs, highly flexible to change (new platforms/architectures -*HyDe*)
- **Weaknesses**
    - The performance could be further improved, might need enhancements for more complex architectures, the output of the system can be improved
- **Opportunities**
    - Take advantage of other more complex AI/ML algorithms, analysis of the body of the functions, build tools for education and industry
- **Threats**
    - More experiments should be run to strengthen our findings, different platforms and languages might uncover issues or new corner cases

Contributions

1. **A deep examination of the context**
    1.1 Building a better understanding of the importance of software architecture among students, instructors, and mobile developers by analyzing their answers to function-specific questionnaires [1]
    1.2 Conducting an overview and comparison of the software architectures used in building mobile applications [2]
    1.3 Analysis of the most common architectural issues that are present at each architectural layer found in the codebases of mobile applications that implement the MVC architectural pattern [3]
2. **Definition and formalization of the problem**
    2.1 The formalization of the problem of automatically inferring architectural layers from mobile codebases
    2.2 Constructing the benchmarks (ground-truth for the codebases) for testing and validating our approaches on both open-source and closed-source applications of different size

Contributions

4. **Different approaches for solving the problem of automatically inferring architectural layers from mobile codebases**
    4.1 Three different approaches for inferring software architecture layers from codebases that use SDKs for building their User Interfaces (UI) [4], [5], [6], [7], [8]
5. **Assessments**
    5.1 A comparison of the three approaches on a set of benchmark mobile applications [9], [10], [6], [7], [8]
    5.2 Evaluation of the proposed approaches from the perspective of ground-truth, intrinsic (without ground-truth), and extrinsic through developers' interviews [6], [7], [8]
    5.3 Evaluation of the portability of our approaches on different platforms
    5.4 Conducting a developer survey [11]

Further work - Short term

- More experiments should be run for all the presented approaches
- In the case of *CARL* and *HyDe*, we plan to study feature detection algorithms
- Running *CARL* and *HyDe* on different platforms
- The empirical evaluation should also be expanded to include more developers
- The portability of our solutions should be better studied
- Study approaches for functional programming

Further work - Long term

- Fully develop the software architecture check tool
- Since the results for non-mobile platforms also look promising, we plan to expand the checker tool to more platforms
- We would like to also integrate our tool into open-source IDEs such as Eclipse
- Last but not least, we would like to introduce new features in the architecture checker tool, such as recommending an architectural change or analyzing the evolution of projects

Questions

References i

- D. Dobrean and L. Dioşan, “Importance of software architectures in mobile projects.,” in *Proceedings of the IEEE 15th International Symposium on Applied Computational Intelligence and Informatics*, 2021.
- D. Dobrean and L. Dioşan, “A comparative study of software architectures in mobile applications.,” *Studia Universitatis Babes-Bolyai, Informatica*, vol. 64, no. 2, 2019.
- D. Dobrean and L. Dioşan, “Model View Controller in ios mobile applications development,” pp. 547–552, KSI Research Inc. and Knowledge Systems Institute Graduate School, 2019.
- D. Dobrean, “Automatic examining of software architectures on mobile applications codebases,” in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 595–599, IEEE, 2019.
- D. Dobrean and L. Dioşan, “Avenues for mining the model-view-controller software architecture on mobile applications,” *Soft Computing – submitted*.
- D. Dobrean and L. Dioşan, “An analysis system for mobile applications MVC software architectures,” pp. 178–185, INSTICC, SciTePress, 2019.

References ii

- D. Dobrean and L. Dioşan, “Detecting model view controller architectural layers using clustering in mobile codebases,” pp. 196–203, INSTICC, 2020.
- D. Dobrean and L. Dioşan, “A hybrid approach to mvc architectural layers analysis,” 2021.
- D. Dobrean and L. Dioşan, “Comparing automatic approaches for mvc architecture detection in ios codebases,” *Automated Software Engineering – submitted*.
- D. Dobrean and L. Dioşan, “On what types of applications can clustering be used for inferring mvc architectural layers?,” *The 24th International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems – submitted*.
- C. H. D. D. Dave Vewer, Benjamin Hendricks, “ios developer survey.” link, 2019-2020.