On what types of applications can clustering be used for inferring MVC architectural layers?

Ph.D. **Dragoş Dobrean**,
Professor Ph.D. **Laura Dioşan**

September 1, 2022

Computer Science Department, **Babes Bolyai University**, Cluj Napoca, Romania

Table of contents

1. Introduction & Context
2. Model View Controller (MVC)
3. Approach
4. Analysis
5. Findings & Future work

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]

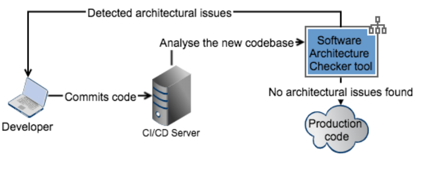
[1] Sarah Perez, **Consumers now average 4.2 hours per day in apps, up 30% from 2019**, TechCrunch, April 8, 2021

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)

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
No architectural issues found
Developer
CI/CD Server
Production code

Mobile architecture checker system in a CI/CD pipeline [2]

[2] Dobrean, D., **Automatic Examining of Software Architectures on Mobile Applications Codebases**, (IEEE International Conference on Software Maintenance and Evolution (pp. 595-599))

Model View Controller (MVC)

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 [3]
- **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

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

Approach

Clustering ARchitecture Layers (*CARL*) [4]

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

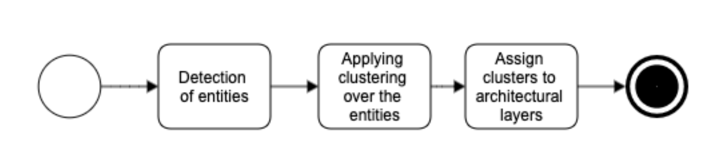
[4] Dobrean, D., Dioşan, L. **Detecting Model View Controller Architectural Layers using Clustering in Mobile Codebases**, (Proceedings of the 15th International Conference on Software Technologies (2020), pages 196-203)

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

CARL – Feature selection – approaches

- **F1 - Number of dependencies** – how many dependencies does a component have with all the other components
- **F2 - Presence of dependencies** – if a dependency between two codebase elements is present
- **F3 - Name distance** – F2 + distances between the name of the components
- **F4 - Keywords presence** – F3 + presence of a keyword (view, controller)
- **F5 - SDK Inheritance** – F4 + SDK inheritance of the component

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 – Feature selection

	Model		View		Controller		Accu-
	Precision	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**	**racy**
CARL-F_1	0.50	0.01	0.22	1,00	1,00	0.10	0.24
CARL-F_2	0.49	0.93	0.17	0.09	1,00	0.08	0.46
CARL-F_3	0.62	0.75	0.33	0.53	0.65	0.22	0.52
CARL-F_4	0.70	0.93	0.84	0.83	0.99	0.56	0.78
CARL-F_5	0.76	0.99	1,00	1,00	0.99	0.57	0.85

Analysis of all the five versions of CARL on the benchmark application

Analysis - Detection quality

Codebase	Model		View		Controller		Accu-
	Precision	Recall	Precision	Recall	Precision	Recall	racy
Firefox	0.92	0.95	1.00	0.99	0.73	0.64	0.91
Wikipedia	0.78	0.83	1.00	0.54	0.83	0.98	0.82
Trust	0.79	0.69	0.38	0.66	0.62	0.57	0.66
E-comm	0.76	0.99	1.00	1.00	0.99	0.57	0.85
Game	0.87	0.95	0.75	1.00	1.00	0.75	0.88
Stock	0.64	0.98	1.00	0.59	1.00	0.61	0.76
Education	0.55	0.98	0.50	0.05	0.95	0.44	0.62
Demo	0.96	1.00	1.00	0.75	1.00	1.00	0.96

CARL-F_5 results in terms of detection quality

Analysis - ML Metrics

Approach	Size	Model		View		Controller		Average		Accuracy	Homog.	Compl.
		Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall			
CARL F_5	Small	0.87	0.95	0.75	1.00	1.00	0.75	0.87	0.90	0.93	0.77	0.84
CARL F_5	Medium	0.66	0.93	0.83	0.39	0.93	0.68	0.81	0.67	0.74	0.35	0.45
CARL F_5	Large	0.82	0.88	0.79	0.88	0.78	0.59	0.80	0.78	0.81	0.48	0.51

Average (on applications classes) precision, recall, accuracy, Homogeneity, and Completeness of the analyzed codebases against the ground truth. Note that precision and recall metrics are computed both at layer-level (columns *Model, View and Controller*) and at codebase-level, as a mean over all three layers (column *Average*).

Analysis - Clustering

Approach	Size	Adjusted Rand Index	Mean Silhouette coefficient	Davies Bouldin index
CARL F_5	Small	0.80	0.92	0.18
CARL F_5	Medium	0.33	0.78	0.37
CARL F_5	Large	0.50	0.76	0.40

Average (on applications classes) Adjusted Rand Index, Mean Silhouette Coefficient, and Davies Bouldin Index.

Findings & Future work

Threats to validity

- Feature selection based on a trial and error approach
- iOS platform, Swift language
- MVC only
- More experiments should be run

Conclusions

- Increased the confidence in applying AI to the field of software engineering on mobile platforms
- *CARL* works well in small and large-sized codebases that respect best practices
- Our method is unsupervised, requires no prior knowledge
- Paves the way for automatic architectural detection of mobile codebases

Further work

- More experiments on different-sized codebases
- Study feature detection algorithms
- Study approaches for functional programming

Questions