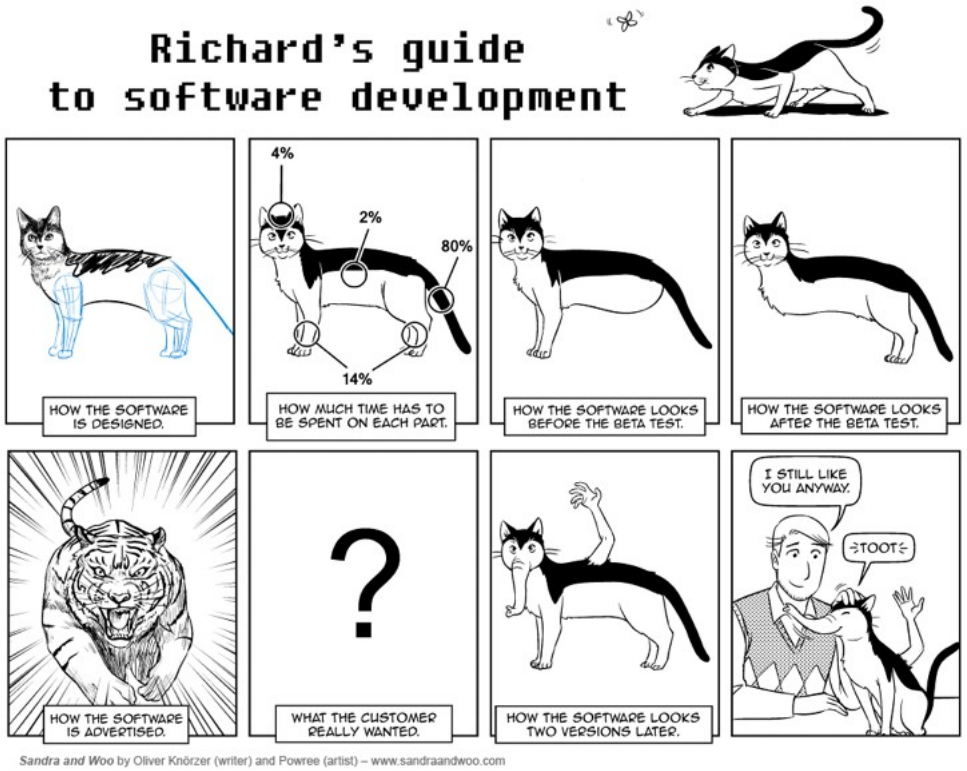
Empirical software engineering

Ivano Malavolta



VU VRIJE UNIVERSITEIT AMSTERDAM

LOOKING FURTHER

Roadmap

Empirical software engineering

- definition
- dimensions
- types of empirical strategies

Empirical software engineering

Scientific use of quantitative and qualitative data to

- **understand and**
- **improve**

software products and software development processes

[Victor Basili]

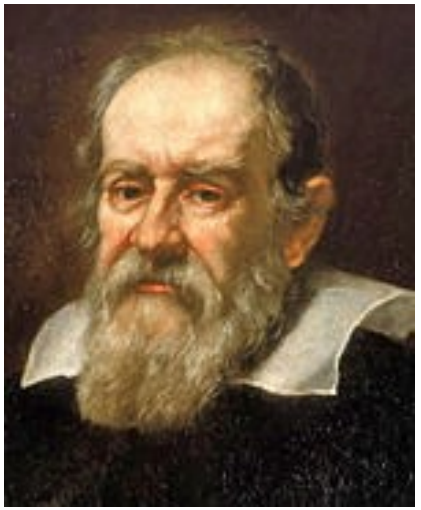
Data is central to address any research question

Issues related to **validity** and **replicability** addressed

Intuition

It is an application of the **scientific method**

- ask yourself a *question*
- background *research*
- formulate an *hypothesis*
- setup an *experiment*
- observe *phenomenon*
- perform *analysis* on your results
- draw *conclusions*



Why empirical studies?

Anecdotal evidence or “common-sense” often not good enough

- Anecdotes often insufficient to **support decisions** in the industry
- Professionals need better advices on **how** and **when** to use methodologies

Evidence important for successful **technology transfer**

- Systematic gathering of evidence
- Wide dissemination of evidence

Dimensions of empirical studies

"In the lab" versus "in the wild" studies

Quantitative versus qualitative studies

“In the lab” VS “in the wild” studies

Common “in the lab” methods

- Controlled experiments
- Literature reviews
- Simulations

Common “in the wild” methods

- Quasi-experiments
- Case studies
- Survey research
- Ethnographies
- Action research

Quantitative VS qualitative studies

Quantitative research

quantifying a relationship or to compare two or more groups with the aim to identify a cause-effect relationship

- fixed implied factors
- focus on quantitative data → promotes comparison and statistical analyses

Qualitative research

studying objects in their natural setting and letting the findings emerge from the observations

- inductive process (usually discovers the “why”)
- the subject is the person

Example of quantitative study

2017 IEEE International Conference on Software Maintenance and Evolution

An Empirical Study of Local Database Usage in Android Applications

Yingjun Lyu*, Jiaping Gui*, Mian Wan, William G.J. Halfond
Department of Computer Science
University of Southern California
Email: {yingjunl, jgui, mianwan, halfond}@usc.edu

***Abstract*—Local databases have become an important component within mobile applications. Developers use local databases to provide mobile users with a responsive and secure service for data storage and access. However, using local databases comes with a cost. Studies have shown that they are one of the most energy consuming components on mobile devices and misuse of their APIs can lead to performance and security problems. In this paper, we report the results of a large scale empirical study on 1,000 top ranked apps from the Google Play app store. Our results present a detailed look into the practices, costs, and potential problems associated with local database usage in deployed apps. We distill our findings into actionable guidance for developers and motivate future areas of research related to techniques to support mobile app developers.**

***Index Terms*—Mobile applications, database, energy, performance, security, empirical study.**

I. Introduction

The past decade has seen incredible growth in the amount of mobile apps available to end users. Developers compete for end users by designing innovative apps that can provide a rich user experience by combining data from web services, location services, and sensor data. In addition to innovating new services, developers also strive to create apps that are

For example, both local and remote databases can be attacked using SQL injection (SQLI) attacks, if developers do not follow best practices for validating user input and interacting with the databases [4], [5], [6]. However, some of the problems are unique to the context in which mobile devices operate. Many database operations, such as transactions, require file locking and rollback capability. This means that the use of such operations is resource intensive, requiring CPU time, memory, and I/O access. Unsurprisingly, the use of these operations means that the use of local databases also comes with a high energy cost; recent studies have found that they are among the top two or three energy consumers of all services on mobile devices [7]. Lastly, despite the seemingly similar use of SQL, local databases often have different query syntaxes and semantics than traditional remote databases. This can mean that typical operations, such as batching multiple database queries, are handled differently on local and remote databases, potentially causing apps to behave incorrectly. Developers lack support for identifying when their code may suffer from these problems (e.g., SQLI and syntactical issues) and performance information (e.g., energy) that can help them design and implement more efficient apps.

Example of qualitative study

Migrating towards Microservice Architectures: an Industrial Survey

Paolo Di Francesco
Gran Sasso Science Institute
L'Aquila, Italy
paolo.difrancesco@gssi.it

Patricia Lago
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
p.lago@vu.nl

Ivano Malavolta
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
i.malavolta@vu.nl

***Abstract*—Microservices are gaining tremendous traction in industry and a growing scientific interest in academia. More and more companies are adopting this architectural style for modernizing their products and taking advantage of its promised benefits (e.g., agility, scalability). Unfortunately, the process of moving towards a microservice-based architecture is anything but easy, as there are plenty of challenges to address from both technical and organizational perspectives.**

In this paper we report about an empirical study on migration practices towards the adoption of microservices in industry. Specifically, we designed and conducted a survey targeting practitioners involved in the process of migrating their applications and we collected information (by means of interviews and questionnaires) on (i) the performed activities, and (ii) the challenges faced during the migration. Our findings benefit both (i) researchers by highlighting future directions for industry-relevant problems and (ii) practitioners by providing a reference framework for their (future) migrations towards microservices.

***Index Terms*—Microservice Architecture, Migration to Microservices, Industrial Survey**

I. Introduction

There is no single and rigorous definition about the microservice architecture (MSA) style. The most adopted definition in scientific papers [5], is the one provided by Lewis and Fowler, which describes this style as *an approach for developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API* [8].

The MSA style brings both significant benefits and challenges. If on one hand microservice systems have flexible and evolvable architectures [6], on the other hand many technical challenges (e.g., infrastructure automation, distributed debugging) and organizational challenges (e.g., creation of cross-functional teams) need to be faced before being able to fully benefit from them. Adopting MSAs is challenging [20], either for developing a greenfield system, or for migrating an existing system that suffers from issues that become more and more difficult to solve. Typical issues of legacy systems are technical (e.g., the system becomes highly coupled, hard to maintain, presents side effects) or business-related (e.g., long time to release new features, low productivity of developers). In some cases, migrating towards MSA represents the best option for resolving existing issues and at the same time improving the system maintainability and the frequency of product releases.

In this study we present an in-depth investigation among practitioners to uncover the migration practices towards MSA. To this aim, we study (i) the activities and (ii) the challenges faced by industrial practitioners when migrating towards MSA. The research **methodology** we used relied on two main phases. In the first phase, we designed an interview guide that we used for conducting 5 exploratory interviews with industrial practitioners. In the second phase, we used the interview guide and the results of the interviews to design an online survey questionnaire, which we distributed among our network of practitioners. In total, 18 practitioners across 16 different IT companies and at different professional stages participated to our study. The high-quality data contributed by our participants makes this a rather exciting first study in the state of the practice in migrations towards MSA, and hence an hopefully inspiring food for thought for future research.

The main **contributions** of this paper are the following:

- a survey of 18 practitioners that provides quantitative information about migrations towards MSA;
- an analysis of the collected data that discusses practitioners perspectives on migration activities and challenges in each of the three phases of: (i) architecture recovery of the pre-existing system, (ii) architecture transformation from the pre-existing to the new architecture, and (iii) the implementation of the new system;
- a discussion of the obtained results in terms of practitioners activities and potentially relevant research directions;
- the study replication package[1], featuring the raw data of the online questionnaire.

The rest of the paper is organized as follows. In Section II we describe the migration model for MSA we used throughout the study. In Section III we present the design of the study. In Sections IV, V, and VI we present the results of our survey, followed by our reflections on the topic of migrations towards MSA (Section VII). Threats to validity and related work are described in Sections VIII and IX, respectively. With Section X we close the paper and discuss future work.

II. Migrating towards MSA

Inspired by the *horseshoe model* by Kazman et al. [13], Razavian et al. [23] framed the processes characterizing the

[1] http://www.s2group.cs.vu.nl/icsa-2018-replication-package

Example

A Quantitative and Qualitative Investigation of Performance-Related Commits in Android Apps

Teerath Das*, Massimiliano Di Penta†, Ivano Malavolta‡
*Gran Sasso Science Institute, L'Aquila, Italy - teerath.das@gssi.infn.it
†University of Sannio, Benevento, Italy - dipenta@unisannio.it
‡Vrije Universiteit Amsterdam, The Netherlands - i.malavolta@vu.nl

***Abstract*—Performance is nowadays becoming a crucial issue for mobile apps, as they are often implementing computational-intensive features, are being used for mission-critical tasks, and, last but not least, a pleasant user experience often is a key factor to determine the success of an app. This paper reports a study aimed at preliminarily investigating to what extent developers take care of performance issues in their commits, and explicitly document that. The study has been conducted on commits of 2,439 open source Android apps, of which 180 turned out to contain a total of 457 documented performance problems. We classified performance-related commits using a card sorting approach, and found that the most predominant kinds of performance-related changes include GUI-related changes, fixing code smells, network-related code, and memory management.**

***Index Terms*—Android, Mobile Performance issues, App Store Mining.**

I. Introduction

Mobile applications are nowadays gaining a huge popularity and importance. From a purely economical standpoint, their market is incredibly increasing, and it has been estimated it will reach about $70 billion in annual revenue by 2017 [3]. Besides that, it is possible to observe two phenomenon. For some operating systems—such as Android—the number of available devices is increasing, each one having its specific characteristics, *e.g.*, in terms of CPU, memory, Graphical Processing Unit (GPU), and screen. In general, the hardware market is releasing devices with better and better performance, nowadays comparable to desktop computers. In such a context, it may not be infrequent that an app undergoes fixes with the aim of dealing with performance problems. These problems may be due to the addition of a new feature in a scenario where developers mainly focus on ensuring an early release. Or else, they could happen for an improper usage of the Android APIs on, in general, because of bad design/implementation choices.

So far, performance issues have been investigated in Web applications [1], heterogeneous environments [5], or large-scale applications [14]. Also, recently Zaman *et al.* have conducted a qualitative study of performance bugs [20]. To the best of our knowledge, there is only work on the analysis of mobile app performance bugs by Liu *et al.* [12], limited to the analysis of 70 real bugs.

using regular expressions, commits explicitly referring to performance-related issues. In other words, instead of using static source code analysis—as done by Liu *et al.* [12], and instead of using dynamic analysis—which would require appropriate execution profiles, and not practicable on large-scale—we rely on documented performance related changes, as previously done by Ray *et al.* [17] for the analysis of bug categories on GitHub.

We report the distribution of such commits, also analyzing how do they vary across app categories. Then, using the card sorting approach, we produce a taxonomy of performance related concerns, and qualitatively describe some examples of commits belonging to these concerns. With respect to work such as the one of Liu *et al.* [12], our study has been conducted in the large, featuring the analysis of 35,880 commits from 2,439 open source Android apps. Of such c
457 (1.27%) of them, belonging to 180 apps, turne
be documented, performance-related commits. They g
affected any kinds of apps, although categories in wh
experience was very important—*e.g.*, health and fit
photography—were slightly more affected than oth
card sorting categorization revealed how the most
kinds of performance-related commits were about GU
changes, removing (performance-affecting) code smells, and
dealing with network or memory-related problems.

In summary, the main **contributions** of this paper can be summarized as: (i) results of a study investigating performance-related commits in 180 open source Android apps; (ii) a taxonomy of the main kinds of performance-related problems, obtained by applying card sorting [19]; and (iii) the study replication package, featuring a dataset of categorized performance-related commits[1].

II. Study Design

The *goal* of this study is to investigate performance-related commits in Android apps, with the *purpose* of understanding their nature and their relationship with projects' characteristics, such as project domain or size. The study *context* consists of 2,439 open-source apps and their evolution history. The study aims at addressing the following research questions:

They are complementary

Types of empirical strategies

Experiment

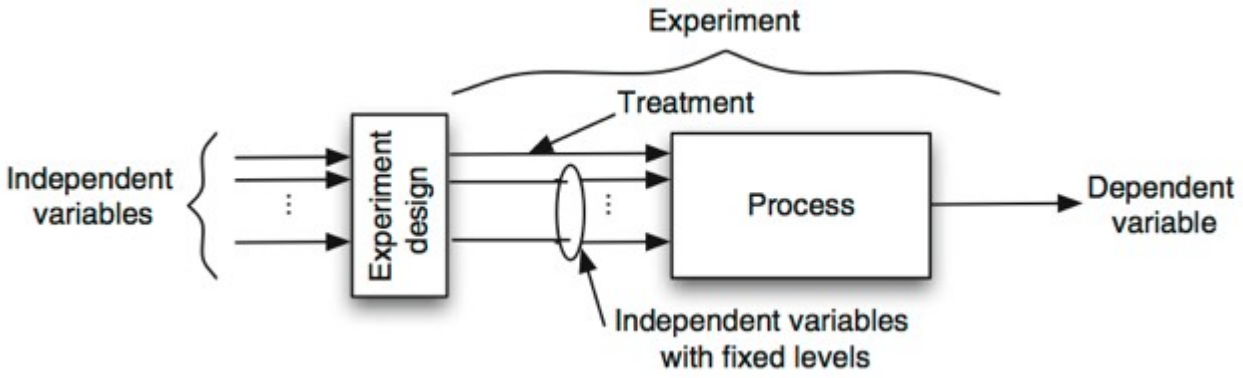
Survey

Case study

Experiment

Def: an empirical enquiry that manipulates one factor or variable of the studied setting

1. Identify and understand the variables that play a role in software development, and the connections between variables
2. Learn cause-effect relationships between the development process and the obtained products
3. Establish laws and theories about software construction that explain development behaviour



A more intuitive definition...

Experiment

Models key characteristics of a reality in a controlled environment and manipulating them iteratively to investigate the impact of such variations and get a better understanding of a phenomenon

Laboratory

Simplified and controllable reality where the phenomenon under study can be manipulated

CONTROL - CONTROL - CONTROL

Example

Assessing the Impact of Service Workers on the Energy Efficiency of Progressive Web Apps

Ivano Malavolta, Giuseppe Procaccianti, Paul Noorland, Petar Vukmirović
Computer Science Department, Vrije Universiteit Amsterdam, The Netherlands
{i.malavolta | g.procaccianti | p.vukmirovic}@vu.nl, p.d.noorland@student.vu.nl

***Abstract*—*Context.* Mobile web apps represent a large share of the Internet today. However, they still lag behind native apps in terms of user experience. Progressive Web Apps (PWAs) are a new technology introduced by Google that aims at bridging this gap, with a set of APIs known as service workers at its core.**
***Goal.* In this paper, we present an empirical study that evaluates the impact of service workers on the energy efficiency of PWAs, when operating in different network conditions on two different generations of mobile devices.**
***Method.* We designed an empirical experiment with two main factors: the use of service workers and the type of network available (2G or WiFi). We performed the experiment by running a total of 7 PWAs on two devices (an LG G2 and a Nexus 6P) that we evaluated as blocking factor. Our response variable is the energy consumption of the devices.**
***Results.* Our results show that service workers do not have a significant impact over the energy consumption of the two devices, regardless of the network conditions. Also, no interaction was detected between the two factors. However, some patterns in the data show different behaviors among PWAs.**
***Conclusions.* This paper represents a first empirical investigation on PWAs. Our results show that the PWA and service workers technology is promising in terms of energy efficiency.**

I. Introduction

Today the total activity on mobile devices like smartphones and tablets accounts for an incredible 67% of the time spent on digital media in the United States [1]. A considerable share of this amount of time is spent on the mobile web (i.e. accessing mobile-optimized websites via a browser on a mobile device), where a growth of 62% in terms of digital time spent has been observed in the last three years [1].

The mobile web is based on web apps conforming to standard languages like HTML5, CSS3, and JavaScript, which offer (among many) the advantage of full application portability across platforms (e.g. Android, Apple). Even if the browser is becoming more and more a fully-fledged software platform (e.g. the HTML5 standard provides APIs for geolocation, accessing the camera, microphone), as of today the mobile web struggles in providing a satisfactory experience to the user, mainly due to the strong dependence on network conditions, the lack of support for push notifications, and so on.

In this context, *Progressive Web Apps* (PWAs[1]) are a new technology advocated by Google as a way of overcoming the above mentioned limitations. The advantages of PWAs are clear when they are compared to classical web apps, for example: they can be launched from an icon in the home screen of the device (like native apps do), they instantly load regardless of the network availability, they support push notifications. At the core of this new technology is the concept of *service worker*, a set of APIs that allows developers to programmatically cache and preload assets and data, manage push notifications, and others. Technically, a service worker is a JavaScript module running in its own thread and providing generic entry points for event-driven background processing (e.g. reaction to the receiving of a push notification).

If on one side service workers have been advertised as performance boosters, network savers, and providers of better user experience, on the other side they are additional code to be downloaded, parsed, and run by the browser. Under this perspective, it is interesting to investigate the price that web developers and users may have to pay for those features in terms of other software quality aspects, e.g. energy efficiency (battery usage), performance, code complexity.

The **goal** of this paper is to assess the impact of service workers on the *energy efficiency* of PWAs. In this study we focus on the energy efficiency of PWAs because (i) energy is one of the most scarce resources in mobile devices [8] and (ii) there is no evidence about how service workers internally use battery-draining resources like the network. In order to achieve the aforementioned goal, we designed, executed, and reported an empirical study on the energy efficiency of 7 real PWAs running with and without service workers, under different network conditions (2G and WiFi), and on different mobile devices (i.e., low-end and high-end). *To the best of our knowledge, this paper is the first empirical investigation involving service workers and progressive web apps.*

The main **contributions** of this paper are:

- the results of an experiment on the energy efficiency of 7 real progressive web apps under different conditions;
- a discussion of the obtained results and their implications;
- the experiment replication package with all the measured PWAs, the analysis scripts, and raw data.

The **target audience** of this paper is composed of mobile web developers, browser vendors, and researchers. We support mobile web developers in knowing how the service workers technology can impact the energy efficiency of their PWAs; we aim to provide browser vendors with objective evidence about the impact that service workers may have on the energy efficiency of PWAs; finally, we aim at informing researchers

Video and slides about this article as homework for this week

VU

Case study

Def: an empirical enquiry to investigate one instance (or a small number of instances) of a contemporary software engineering phenomenon within its real-life context, especially when the boundary between phenomenon and context cannot be clearly specified

Observational study

Data collected to track a specific attribute or to establish relationships between different attributes

Multivariate statistical analysis is often applied

Example

From 6.2 to 0.15 seconds – an Industrial Case Study on Mobile Web Performance

Jasper van Riet
Vrije Universiteit Amsterdam
The Netherlands
j.c.a.van.riet@student.vu.nl

Flavia Paganelli
30MHz
The Netherlands
flavia@30mhz.com

Ivano Malavolta
Vrije Universiteit Amsterdam
The Netherlands
i.malavolta@.vu.nl

***Abstract*—*Background*. A fast loading web app can be a key success factor in terms of user experience. However, improving the performance of a web app is not trivial and requires a deep understanding of both the browser engine and the specific usage scenarios of the web app under consideration.**
***Aims*. This paper presents an industrial case study targeting a large web-based dashboard, where its performance was improved via 13 distinct interventions over a four-month period.**
***Method*. Firstly, we design a replicable performance engineering plan, where the technical realization of each intervention is reported in details together with its development effort. Secondly, we develop a benchmarking tool which supports 11 widely-used web performance metrics. Finally, we use the benchmarking tool to quantitatively evaluate the performance of the target web app.**
***Results*. We observe a considerable performance improvement over the course of the 13 interventions. Among others, we achieve a 97.56% reduction of the time for the First Contentful Paint (*i.e.,* from 6.29 to 0.15 seconds) and a 19.85% improvement of the Speed Index metric (*i.e.,* from 15.31 to 12.27 seconds).**
***Conclusions*. This case study shows the value of a continuous focus on performance engineering in the context of large-scale web apps. Moreover, we recommend developers to carefully plan their performance engineering activities since different interventions require different efforts and can have very different effects on the overall performance of the system.**

***Index Terms*—Performance, Web Apps, Industrial Case Study.**

I. Introduction

By 2025, 1.2 billion *more* people will be using mobile internet[1]. One of the key enabler of this massive trend is the standardization and the compatibility of Web technologies, primarily lead by the efforts of foundations like W3C and companies like Google and Apple. Indeed, with standard APIs for geolocation, push notifications, accessing the camera, etc., the Web is becoming a fully-fledged software platform and capable of providing richer experiences to end users [6]. The *performance* of a mobile web app is vital towards its success, specially on mobile devices where hardware and connectivity are constrained. For instance, the BBC lost an additional 10% of users for every additional second their web app took to load[2].

However, even though the value of fast web apps is without doubts, improving the performance of mobile web apps is still a very demanding engineering effort. Indeed, today developers need to have a deep understanding of a variety of optimization techniques defined at different levels of abstraction and requiring different technical backgrounds (*e.g.,* caching, prefetching, image optimization, source code bundling) [13]. In this context, assessing how state-of-the-art performance techniques are applied in real industrial projects is fundamental for better understanding their characteristics, gains, and the effort required for their concrete application.

This paper presents an industrial case study we performed at 30MHz, a technology company in the agricultural sector. 30MHz is a scale-up of 35 employees that service over 300 customers across 30 countries. The main product of 30MHz is a web-based dashboard for providing growers insights into geographically-distributed data produced by sensors of humidity, microclimate, wind speed, moisture, etc. With most of the data being produced in real-time by a multitude of networked sensors, the performance of the dashboard is key for the overall experience delivered to end users; and thus a major factor for the success of the business proposition of 30MHz.

The case study is composed of three main phases. In the first phase we design a **performance engineering plan** (PEP) for the 30MHz dashboard, where the technical realization of 13 interventions is reported in detail and with replicability in mind. Secondly, we develop a **benchmarking tool** supporting 11 widely-used web performance metrics such as First Contentful Paint, Median Time to Widget, etc. The benchmarking tool is based on Google Lighthouse[3], an open-source audit tool widely used both in academia and industry. Thirdly, over a period of 4 months we (i) implement each intervention of the performance engineering plan and (ii) apply the benchmarking tool for a **quantitative evaluation** of the performance of the 30MHz dashboard according to the 11 supported metrics.

Thanks to the incremental interventions of the PEP, in this case study we achieved considerable performance improvements across many performance metrics. Among others, we achieve a 97.56% reduction of the time for the First Contentful Paint (from 6.29 to 0.15 seconds) and a 19.85% improvement of the Speed Index metric (from 15.31 to 12.27 seconds). Moreover, we provide an estimation of the engineering effort

[1] https://www.gsma.com/mobileeconomy
[2] https://link.medium.com/OXxPF5BqN6
[3] https://developers.google.com/web/tools/lighthouse

Survey

Def: a system for collecting information from or about **people** to describe, compare or explain their knowledge, attitudes and behavior

Often an investigation performed in retrospect

Interviews and **questionnaires** are the primary means of gathering qualitative or quantitative data

Surveys are done by taking a sample which is representative from the population to be studied

Example

What Industry needs from Architectural Languages: A Survey

Ivano Malavolta, Patricia Lago, Henry Muccini, Patrizio Pelliccione, Antony Tang

Abstract—Many times we are faced with the proliferation of definitions, concepts, languages and tools in certain (research) topics. But often there is a gap between what is provided by existing technologies, and what is needed by their users. The strengths, limitations and needs of the available technologies can be dubious.
The same applies to software architectures, and specifically to languages designed to represent architectural models. Tens of different architectural languages have been introduced by the research and industrial communities in the last two decades. However, it is unclear if they fulfill the user's perceived needs in architectural description. As a way to plan for next generation languages for architectural description, this study analyzes practitioners' perceived strengths, limitations and needs associated to existing languages for software architecture modeling in industry. We run a survey by interviewing 48 practitioners from 40 different IT companies in 15 countries. Each participant is asked to fill in a questionnaire of 51 questions. By analyzing the data collected through this study, we have concluded that (a) whilst practitioners are generally satisfied with the design capabilities provided by the languages they use, they are dissatisfied with the architectural language analysis features and their abilities to define extra-functional properties; (b) architectural languages used in practice mostly originate from industrial development instead of from academic research; (c) more formality and better usability are required of an architectural language.

Index Terms—Software Architecture, Architecture Description Languages, ADL, Survey, Empirical Study

1 Introduction

1.1 The Problem

In their seminal paper on software architecture [1], Dewayne Perry and Alexander Wolf foresee the 90s as the decade of software architecture, and justify the need of a (software) architecting discipline on the conjecture that the slow progress in the development of software systems is due to the excessive focus on development and the limited focus on architecting. Twenty years later, we recognize the impact that software architecture has been having on both academic and industrial worlds. Software architectures are nowadays used for different purposes, including documenting and communicating design decisions and architectural solutions [2], driving analysis techniques (like testing, model and consistency checking, performance analysis [3], [4], [5], [6]), for code generation purposes in model-driven engineering [7], [8], for product line engineering [9], for risks and cost estimation [10], [11], and many more.

One principal issue is the proliferation of languages for software architectures (SA) description without a clear understanding of their merits and limitations. Tens of architectural languages (ALs)[1] can be found today, each characterized by slightly different conceptual architectural elements, different syntax or semantics. They focus on a generic or a specific operational domain; some do not provide design analysis whilst others support different analysis techniques. As observed in [12], one of the reasons for the accumulation of so many architectural languages is the need to satisfy different *stakeholder concerns*: a language has to adequately capture design decisions judged fundamental by the system's stakeholders. Stakeholder concerns are various, ever evolving and adapting to changing system requirements. Hence it is difficult to capture all such concerns with a single, narrowly focused notation. Instead of defining a unique language for software architecture specification, we must accept the existence of domain specific languages for SA modeling, allowing an AL to address specific types of stakeholder concerns. The adoption of UML for modeling architectures (e.g., [13], [14]) did not converge into a standard and uniquely identified notation for SA modeling: a number of UML profiles and extensions have been proposed to enhance the modeling for different concerns, thus further proliferates new architectural languages.

In summary, it is clear that an ideal and general purpose AL is not likely to exist [15], [14], [16], [12]. Rather, architectural languages must be able to focus

Strategies VS dimensions VS factors

Strategy	Design type	Qualitative/ quantitative
Survey	Fixed	Mostly qualitative
Case study	Flexible	Both
Experiment	Fixed	Mostly quantitative

Strategy	Execution control	Measurement control	Ease of replication
Survey	No	No	Medium
Case study	No	Yes	Low
Experiment	**Yes**	**Yes**	**High**

What this lecture means to you?

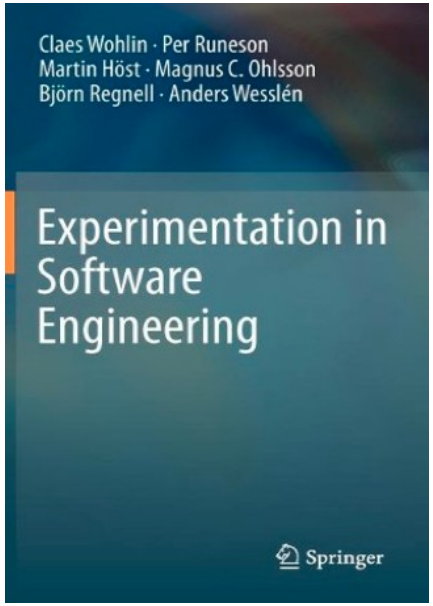
Now you know the basics of **empirical software engineering**

You can make claims that can be eventually **measured**

Many dimensions and types of research in software engineering

- in this course we will focus on controlled experiments

Readings



Chapters 1 and 2

+ IEEE_Software_2021 article on Canvas
+ the MobileSoft_2017 article on Canvas
+ the ICSME_2020 article on Canvas