ICSOFT 2019

14th International Conference on Software Technologies

26 - 28 July, 2019 · Prague, Czech Republic

Quality aspects of serverless architecture: an exploratory study on maintainability

Louis Racicot[1], Nicolas Cloutier[1], Julien Abt[2], Fabio Petrillo[2]

[1]*École Polytechnique de Montréal, Montréal, Canada*

[2]*Université du Québec à Chicoutimi, Chicoutimi, Canada*

{louis.racicot,nicolas.cloutier}@polymtl.com, julien.abt1@uqac.ca,fabio@petrillo.com

UQAC

Département d'informatique
et mathématique

Motivation

- As serverless architecture is fairly new, it is yet to be analyzed as a solution.
- In this paper, we **analyzed the maintainability** of different serverless projects
- Our main research question is:

  Does the serverless impact the maintainability?

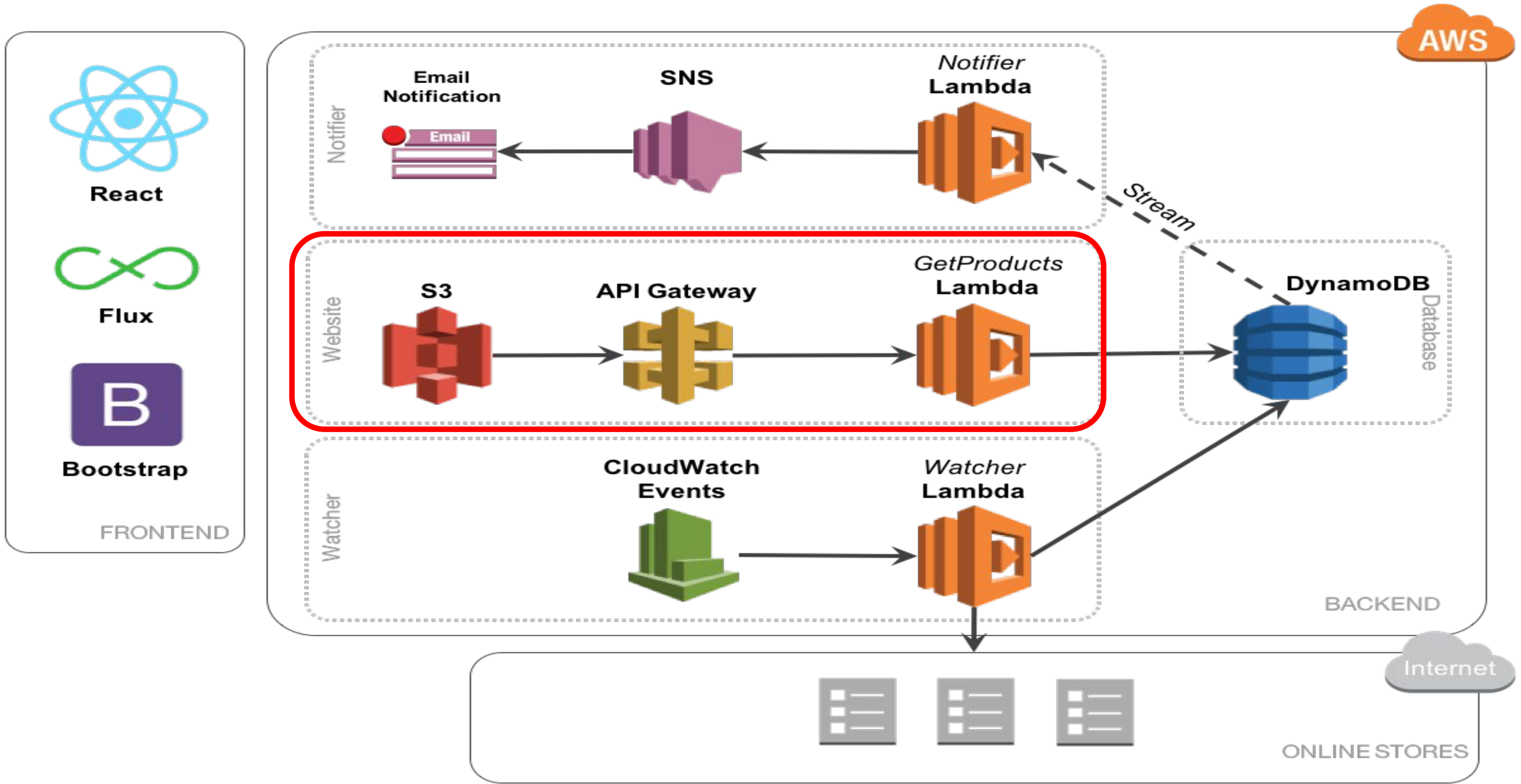
- The purpose of this paper is to identify **maintainability aspects** in serverless based solutions and to uncover those findings for potential future research.

Serverless Architecture

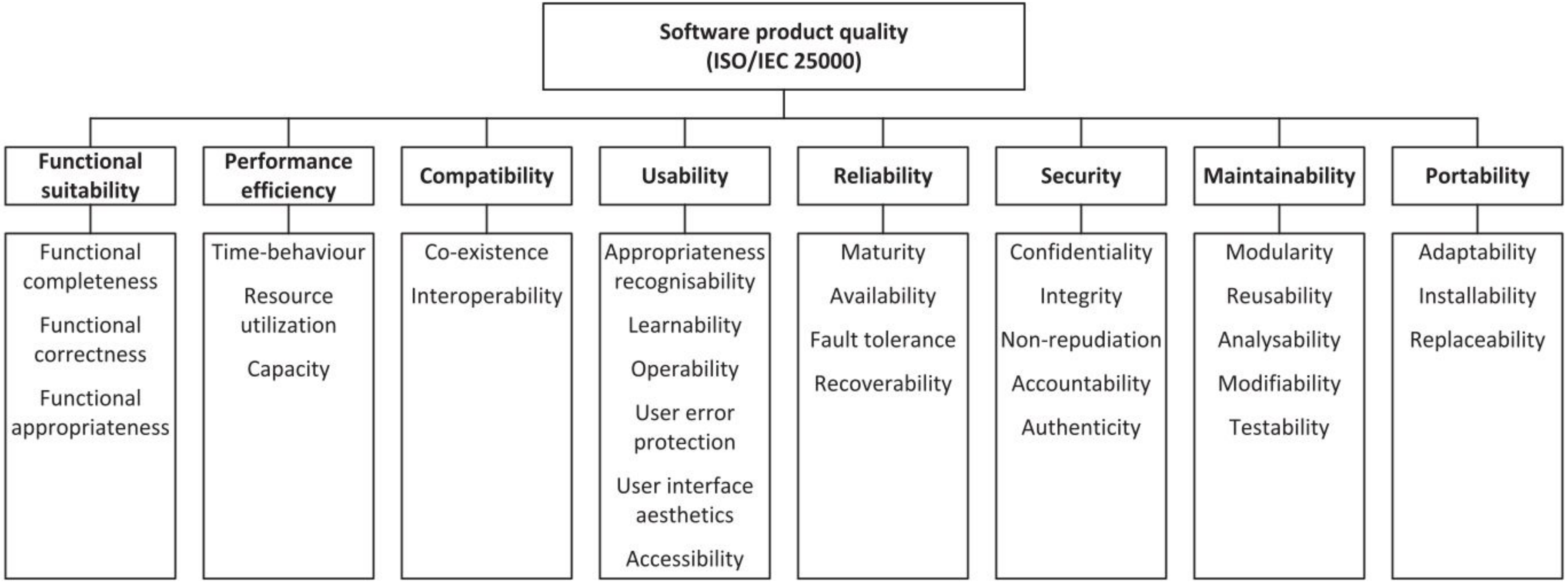
Serverless Architecture

- Serverless architecture is an execution model where the provider dynamically **manage the resources** requested by an application, leasing a server for a **small amount of time** before releasing it for another application to use it.
- The serverless follows a **pay-per-use business model**
- Serverless architecture is emerging more and more popular as the tools are becoming **cheaper** and more accessible
- Designing an architecture presents many advantages especially for **computing intensive and event-driven** applications.

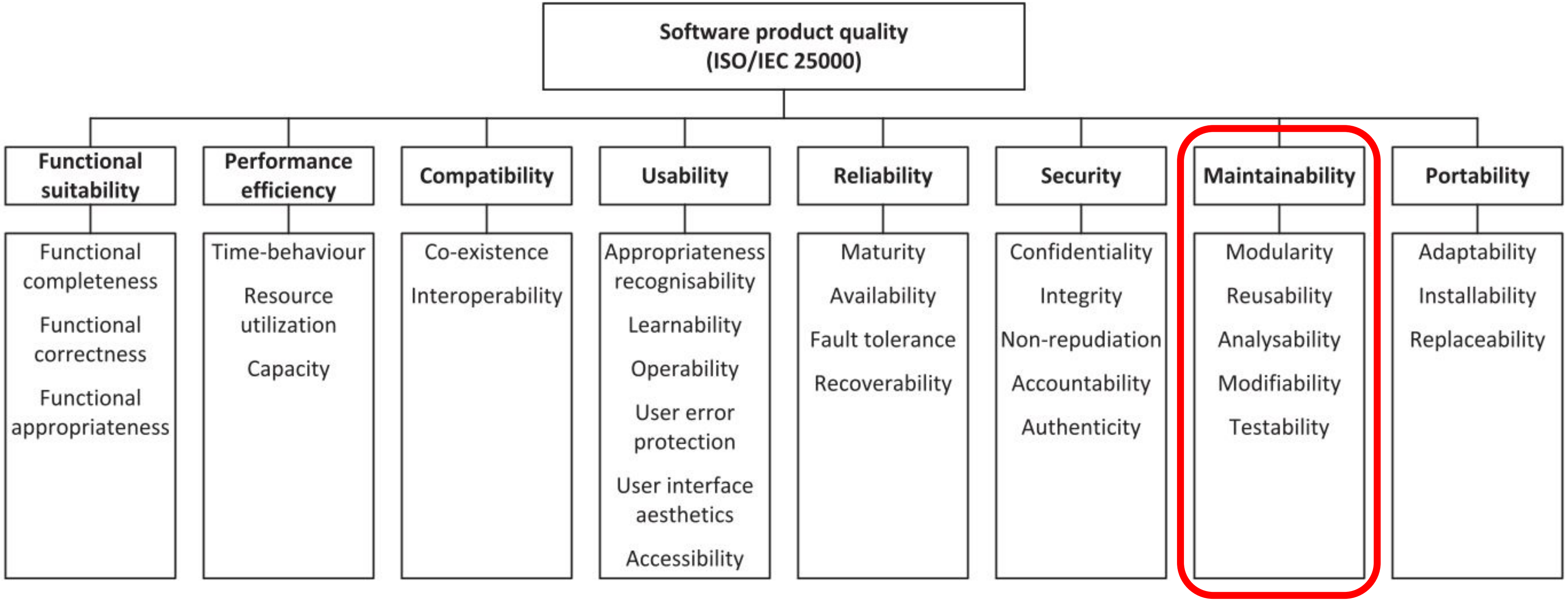
Serverless Architecture



ISO/IEC 25010 - quality model for software products



ISO/IEC 25010 - quality model for software products



Maintainability

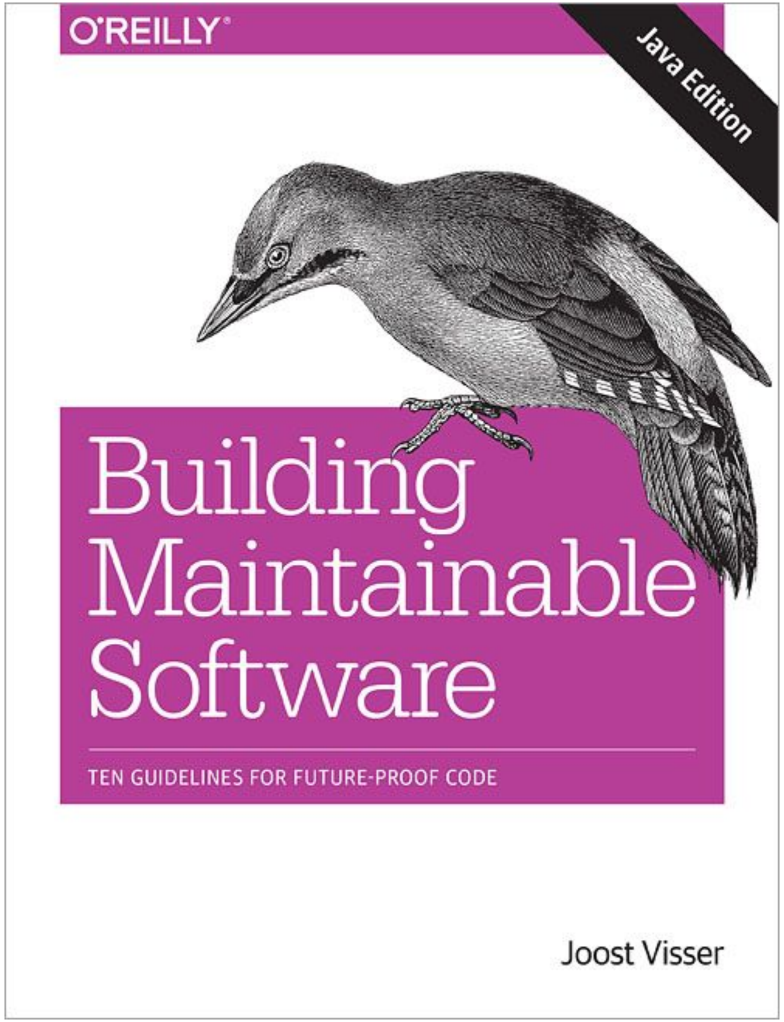
Maintainability [ISO 25010, §4.2.7]

“The degree of ***effectiveness*** *and* ***efficiency*** *with which a product or system can be* ***modified*** *by the intended maintainers”, where modifications can include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications. It also includes installation of updates and upgrades. It can be interpreted as either* ***an inherent capability of the product or system to facilitate maintenance activities****, or the quality in use experienced by the maintainers for the goal of maintaining the product or system.”*

How to determine the quality characteristics of maintainability?

How to determine the quality characteristics of maintainability?

Measuring a set of software product **properties**.


O'REILLY
Java Edition
Building Maintainable Software
TEN GUIDELINES FOR FUTURE-PROOF CODE
Joost Visser

A proposal

***10 guidelines** to help you write source code that is easy to modify.*

"After 15 years of consulting about software quality, we at the Software Improvement Group (SIG) have learned a thing or two about maintainability."

Main principles behind the SIG's 10 guidelines

- Maintainability benefits most from adhering to **simple guidelines**.
- Maintainability is not an afterthought, but should be addressed from the **very beginning** of a development project. **Every individual contribution counts**.
- Some violations are **worse** than others. The more a software system complies with the guidelines, the more maintainable it is.

The 10 SIG's guidelines

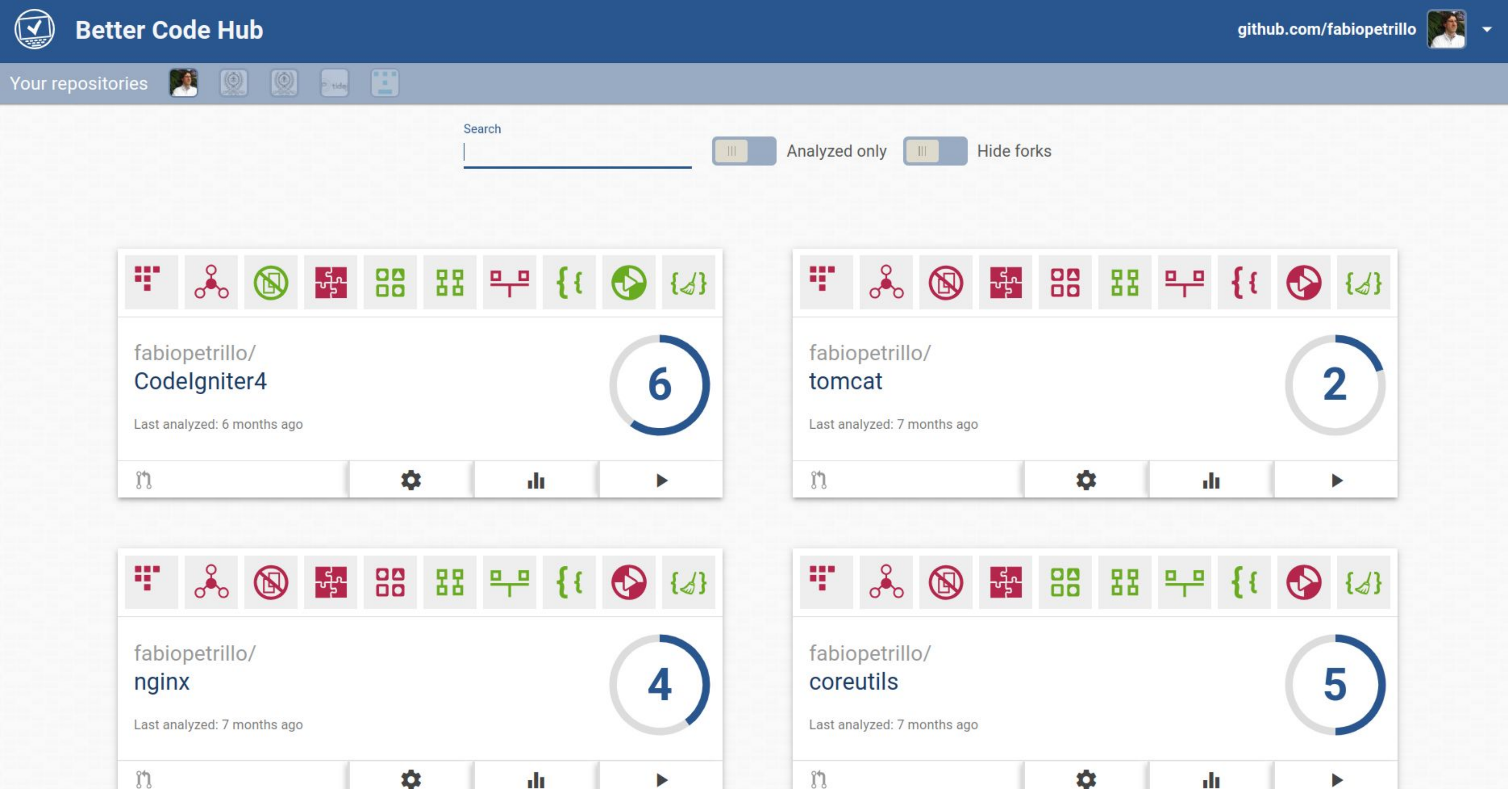
- **Write short units of code**: shorter units (that is, methods and constructors) are easier to analyze, test, and reuse.
- **Write simple units of code**: Units with fewer decision points are easier to analyze and test.
- **Write code once**: duplication of source code should be avoided at all times, since changes will need to be made in each copy. Duplication is also a source of regression bugs.
- **Keep unit interfaces small**: units (methods and constructors) with fewer parameters are easier to test and reuse.
- **Separate concerns in modules**: modules (classes) that are loosely coupled are easier to modify and lead to a more modular system.

The 10 SIG’s guidelines (cont.)

- **Couple architecture components loosely**: top-level components of a system that are more loosely coupled are easier to modify and lead to a more modular system.
- **Keep architecture components balanced**: A well-balanced architecture, with not too many and not too few components, of uniform size, is the most modular and enables easy modification through separation of concerns.
- **Keep your codebase small**: a large system is difficult to maintain, because more code needs to be analyzed, changed, and tested. Also, maintenance productivity per line of code is lower in a large system than in a small system.

The 10 SIG’s guidelines (cont.)

- **Automate development pipeline and tests**: automated tests (that is, tests that can be executed without manual intervention) enable near-instantaneous feedback on the effectiveness of modifications. Manual tests do not scale.
- **Write clean code**: Having irrelevant artifacts such as TODOs and dead code in your codebase makes it more difficult for new team members to become productive. Therefore, it makes maintenance less efficient.



Better Code Hub

github.com/fabiopetrillo

Your repositories

Search

Analyzed only

Hide forks

fabiopetrillo/
CodeIgniter4
6
Last analyzed: 6 months ago

fabiopetrillo/
tomcat
2
Last analyzed: 7 months ago

fabiopetrillo/
nginx
4
Last analyzed: 7 months ago

fabiopetrillo/
coreutils
5
Last analyzed: 7 months ago

Study Design

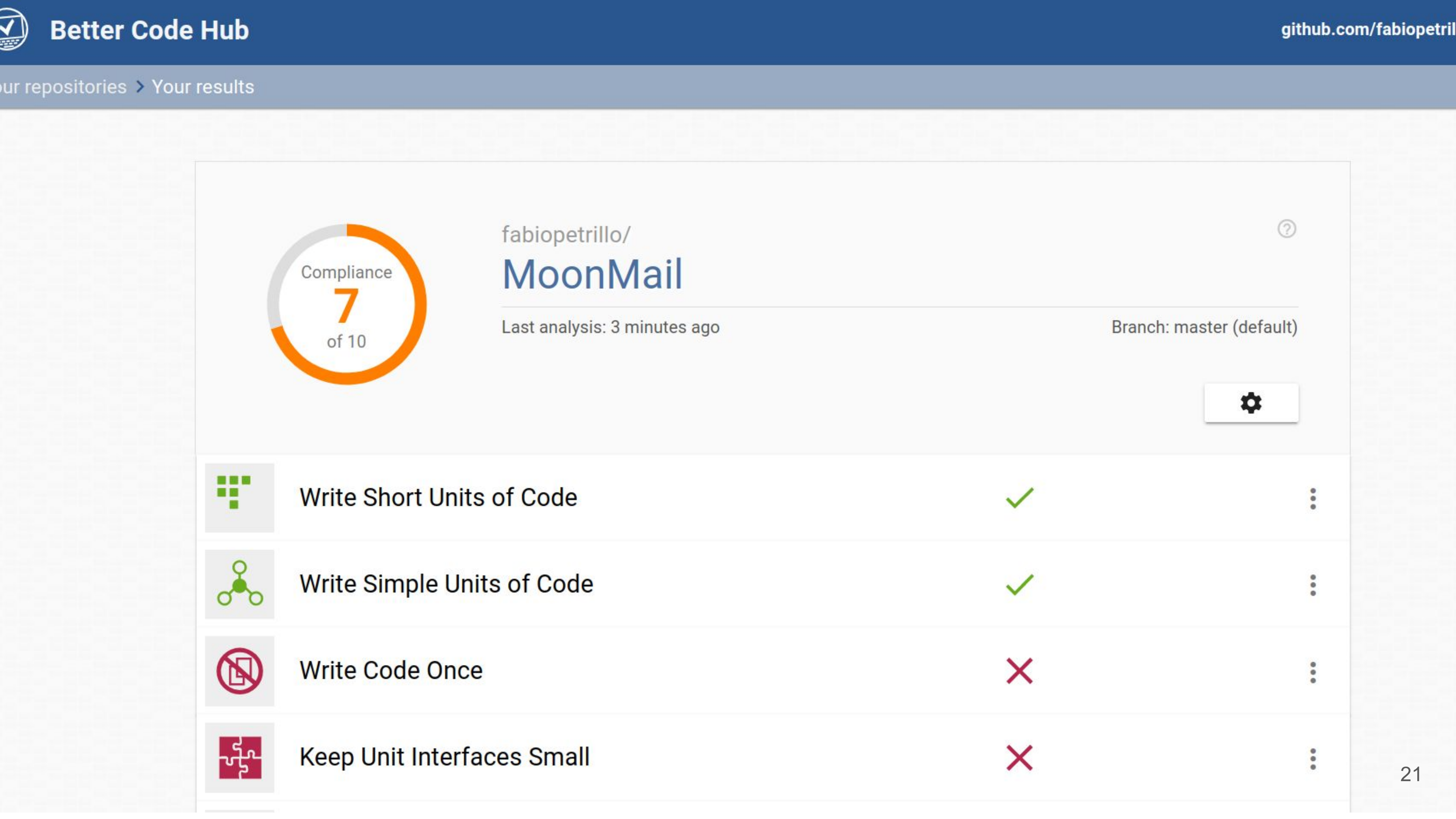
1) Qualitatively discuss possible maintainability and management issues, formulating hypothesis
2) Extract meaningful metrics from FaaS projects to provide empirical results
3) Compare our discussion with empirical results to evaluate the state of maintainability

Our hypotheses

- H1: Software that use serverless technologies have a **low module coupling**, their components are **well balanced and are independent**. Thus, it complies with the modularity aspect of maintainability.
- H2: Software that use serverless technologies have **small units of code with a small interface**.
- H3: Software that use serverless architecture is **easy to analyse**.
- H4: Software that use serverless technologies tends to be **easy to modify** because it has simple units of code and low coupling.
- H5: Software that use serverless technologies are **easy to test** because they have a small code base, they have simple units of code and they are made of independent components.
- H6: The **complexity of deployment** of serverless software is harmful for its maintainability.

Analyzed projects

- To empirically analyze the impacts of FaaS on the code maintainability, we analyzed open-source projects from Github using serverless architecture
- To test these hypotheses, we manually inspected a curated list of serverless projects and we filtered them based on the following criteria:
  - (1) the project must be based on the function-as-a-Service pattern;
  - (2) the project must be of a non-trivial in complexity and in size;
  - (3) the project must be analyzable by BetterCodeHub.
- To extract the metrics, we used **BetterCodeHub**, an online static analysis service whose criteria ares base on guidelines for more maintainable code [Visser 2016].



Better Code Hub

github.com/fabiopetril

ur repositories > Your results

Compliance
7
of 10

fabiopetrillo/

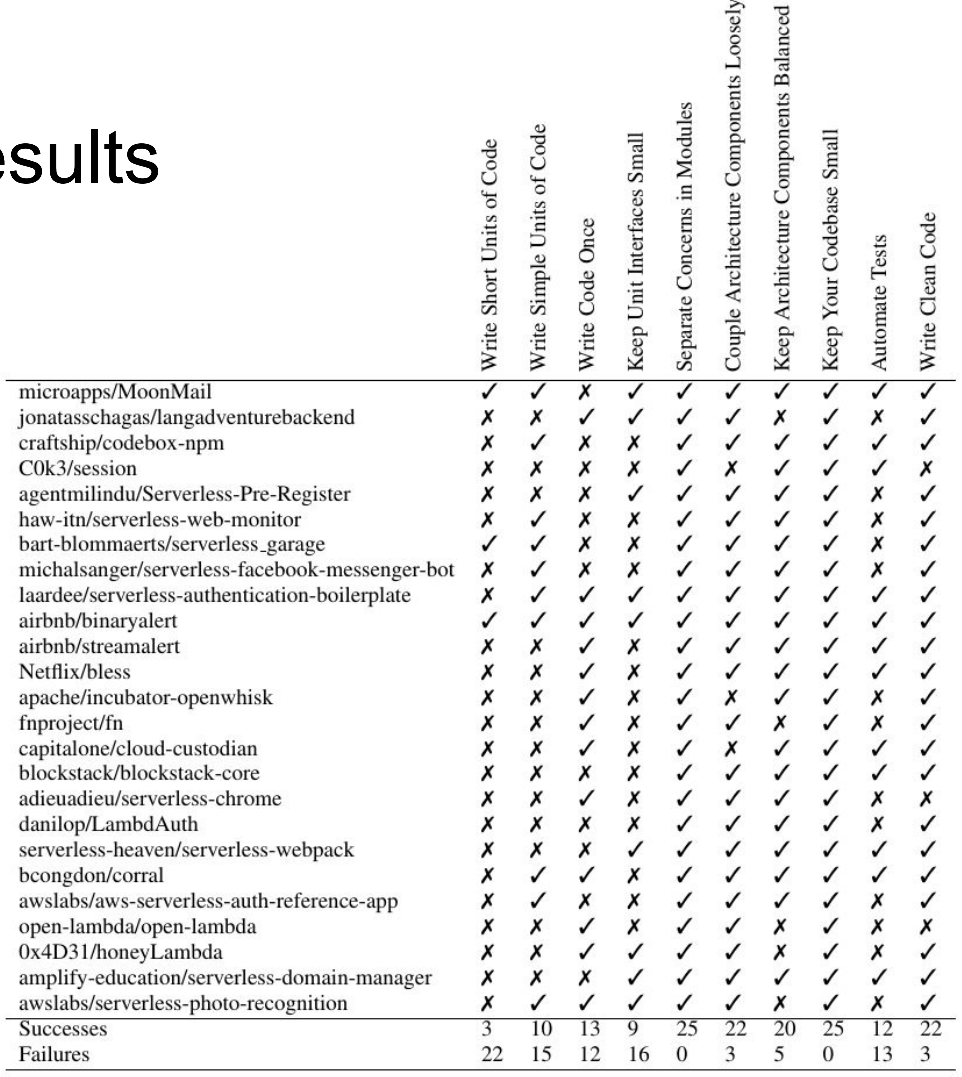
MoonMail

Last analysis: 3 minutes ago

Branch: master (default)

Write Short Units of Code	✓	⋮
Write Simple Units of Code	✓	⋮
Write Code Once	✗	⋮
Keep Unit Interfaces Small	✗	⋮

Results



	Write Short Units of Code	Write Simple Units of Code	Write Code Once	Keep Unit Interfaces Small	Separate Concerns in Modules	Couple Architecture Components Loosely	Keep Architecture Components Balanced	Keep Your Codebase Small	Automate Tests	Write Clean Code
microapps/MoonMail	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓
jonatasschagas/langadventurebackend	✗	✗	✓	✓	✓	✓	✗	✓	✗	✓
craftship/codebox-npm	✗	✓	✗	✗	✓	✓	✓	✓	✓	✓
C0k3/session	✗	✗	✗	✗	✓	✗	✓	✓	✓	✗
agentmilindu/Serverless-Pre-Register	✗	✗	✗	✓	✓	✓	✓	✓	✗	✓
haw-itn/serverless-web-monitor	✗	✓	✗	✗	✓	✓	✓	✓	✗	✓
bart-blommaerts/serverless_garage	✓	✓	✗	✗	✓	✓	✓	✓	✗	✓
michalsanger/serverless-facebook-messenger-bot	✗	✓	✗	✗	✓	✓	✓	✓	✗	✓
laardee/serverless-authentication-boilerplate	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
airbnb/binaryalert	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
airbnb/streamalert	✗	✗	✓	✗	✓	✓	✓	✓	✓	✓
Netflix/bless	✗	✗	✓	✗	✓	✓	✓	✓	✓	✓
apache/incubator-openwhisk	✗	✗	✓	✗	✓	✗	✓	✓	✗	✓
fnproject/fn	✗	✗	✓	✗	✓	✓	✗	✓	✗	✓
capitalone/cloud-custodian	✗	✗	✓	✗	✓	✗	✓	✓	✓	✓
blockstack/blockstack-core	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓
adieuadieu/serverless-chrome	✗	✗	✓	✗	✓	✓	✓	✓	✗	✗
danilop/LambdAuth	✗	✗	✗	✗	✓	✓	✓	✓	✗	✓
serverless-heaven/serverless-webpack	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓
bcongdon/corral	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓
awslabs/aws-serverless-auth-reference-app	✗	✓	✗	✗	✓	✓	✓	✓	✗	✓
open-lambda/open-lambda	✗	✗	✓	✗	✓	✓	✗	✓	✗	✗
0x4D31/honeyLambda	✗	✗	✓	✓	✓	✓	✗	✓	✗	✓
amplify-education/serverless-domain-manager	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓
awslabs/serverless-photo-recognition	✗	✓	✓	✓	✓	✓	✗	✓	✗	✓
Successes	3	10	13	9	25	22	20	25	12	22
Failures	22	15	12	16	0	3	5	0	13	3

25 FaaS Open Source Systems.

Results

	Write Short Units of Code	Write Simple Units of Code	Write Code Once	Keep Unit Interfaces Small	Separate Concerns in Modules	Couple Architecture Components Loosely	Keep Architecture Components Balanced	Keep Your Codebase Small	Automate Tests	Write Clean Code
0x4D31/honeyLambda	✗	✗	✓	✓	✓	✓	✗	✓	✗	✓
amplify-education/serverless-domain-manager	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓
awslabs/serverless-photo-recognition	✗	✓	✓	✓	✓	✓	✗	✓	✗	✓
Successes	3	10	13	9	25	22	20	25	12	22
Failures	22	15	12	16	0	3	5	0	13	3

Results

	Write Short Units of Code	Write Simple Units of Code	Write Code Once	Keep Unit Interfaces Small	Separate Concerns in Modules	Couple Architecture Components Loosely	Keep Architecture Components Balanced	Keep Your Codebase Small	Automate Tests	Write Clean Code
0x4D31/honeyLambda	✗	✗	✓	✓	✓	✓	✗	✓	✗	✓
amplify-education/serverless-domain-manager	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓
awslabs/serverless-photo-recognition	✗	✓	✓	✓	✓	✓	✗	✓	✗	✓
Successes	3	10	13	9	25	22	20	25	12	22
Failures	22	15	12	16	0	3	5	0	13	3

Results

- Our results show that most of analyzed projects separate concerns in modules, couple architecture components loosely, keep architecture components balanced, keep the codebase small, and write clean code
- However **88%** (22/25) of analyzed projects **do not write short units of code**;
- 60% of projects (15/25) write **complex units of code**
- 52% (13/25) do **not use well automated tests**
- 60% (15/25) tend to have **complex unit interfaces**.

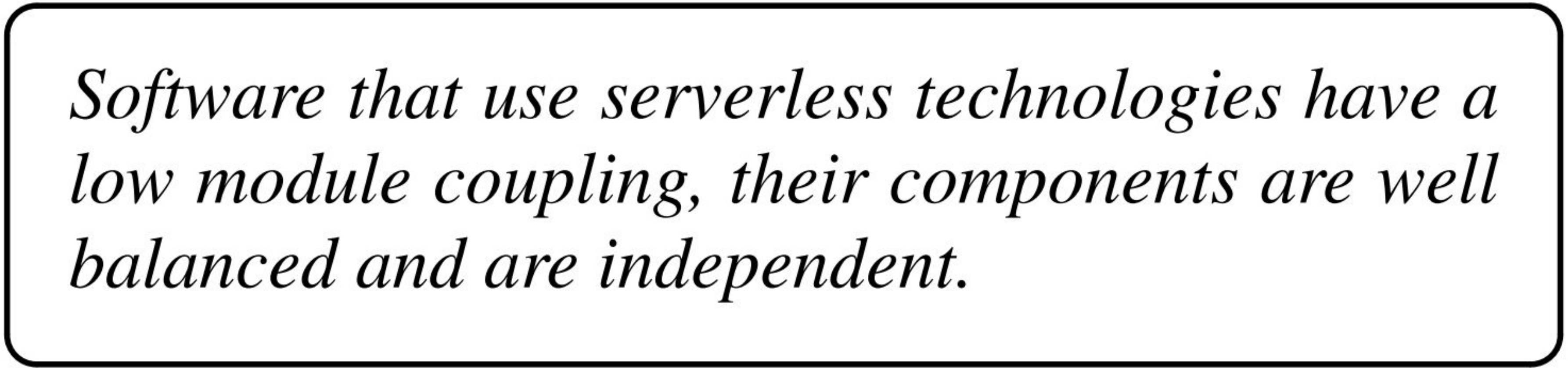
Evaluating our hypotheses

Evaluating our hypotheses

H1: Software that use serverless technologies have a low module coupling, their components are well balanced and are independent. Thus, it complies with the modularity aspect of maintainability.

Evaluating our hypotheses

H1: Software that use serverless technologies have a low module coupling, their components are well balanced and are independent. Thus, it complies with the modularity aspect of maintainability.



Software that use serverless technologies have a low module coupling, their components are well balanced and are independent.

Evaluating our hypotheses

H2: Software that use serverless technologies have small units of code with a small interface.

Evaluating our hypotheses

FALSE

H2: Software that use serverless technologies have small units of code with a small interface.

Serverless projects tend to have big units of code.

Evaluating our hypotheses

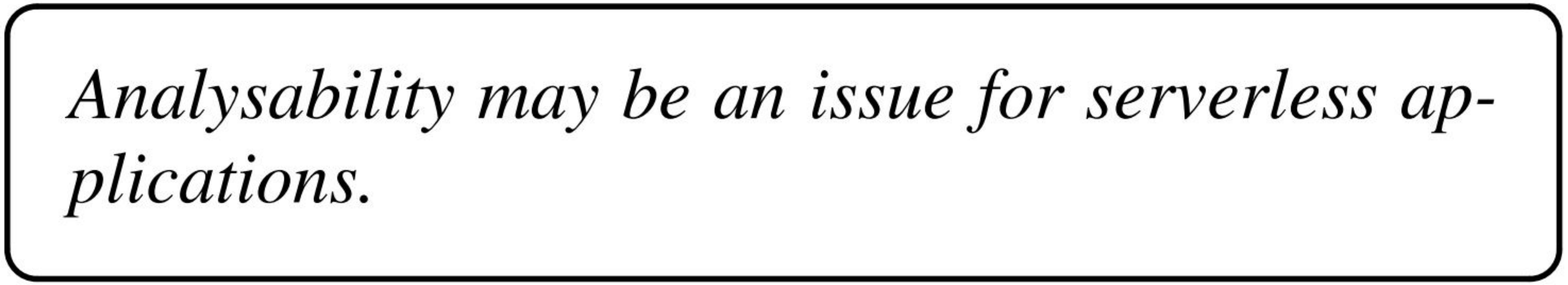
H3: Software that use serverless architecture is easy to analyse.

Evaluating our hypotheses

FALSE

H3: Software that use serverless architecture is easy to analyse.

As expected, the duplication of code is an issue encountered in our analysis. Half the projects encountered this problem. As described in H2, the units of code are big.



Analysability may be an issue for serverless applications.

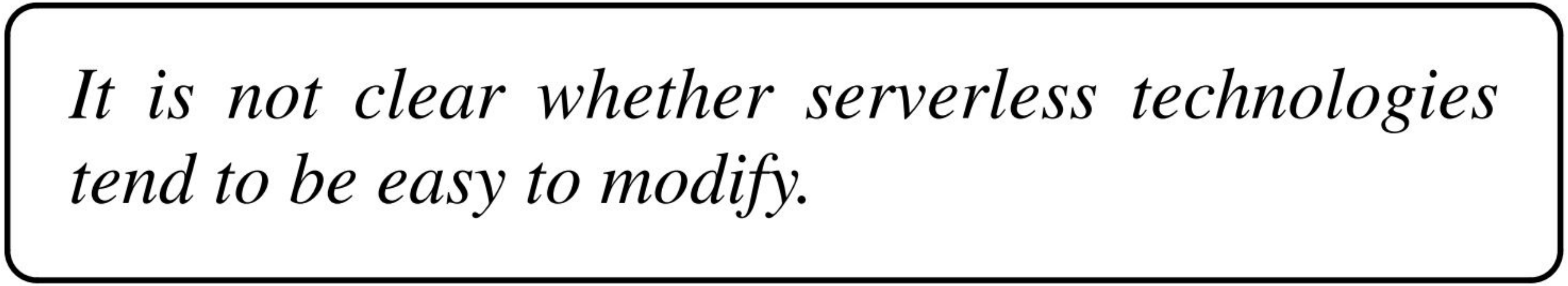
Evaluating our hypotheses

H4: Software that use serverless technologies tends to be easy to modify because it has simple units of code and low coupling.

Evaluating our hypotheses

FALSE

H4: Software that use serverless technologies tends to be easy to modify because it has simple units of code and low coupling.



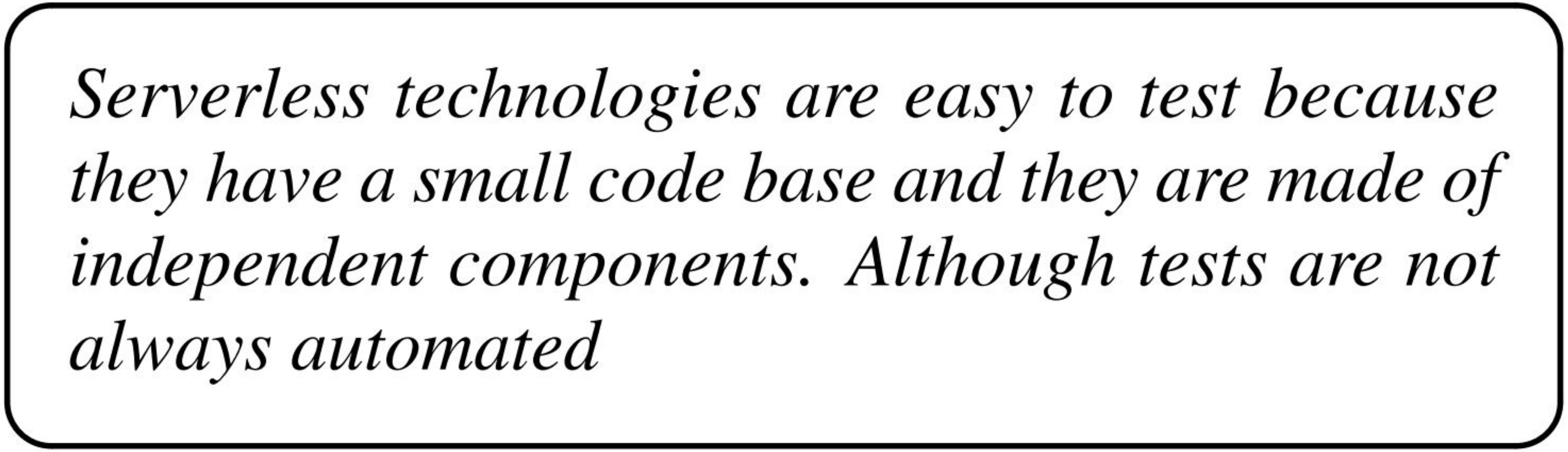
It is not clear whether serverless technologies tend to be easy to modify.

Evaluating our hypotheses

H5: Software that use serverless technologies are easy to test because they have a small code base, they have simple units of code and they are made of independent components.

Evaluating our hypotheses

H5: Software that use serverless technologies are easy to test because they have a small code base, they have simple units of code and they are made of independent components.



Serverless technologies are easy to test because they have a small code base and they are made of independent components. Although tests are not always automated

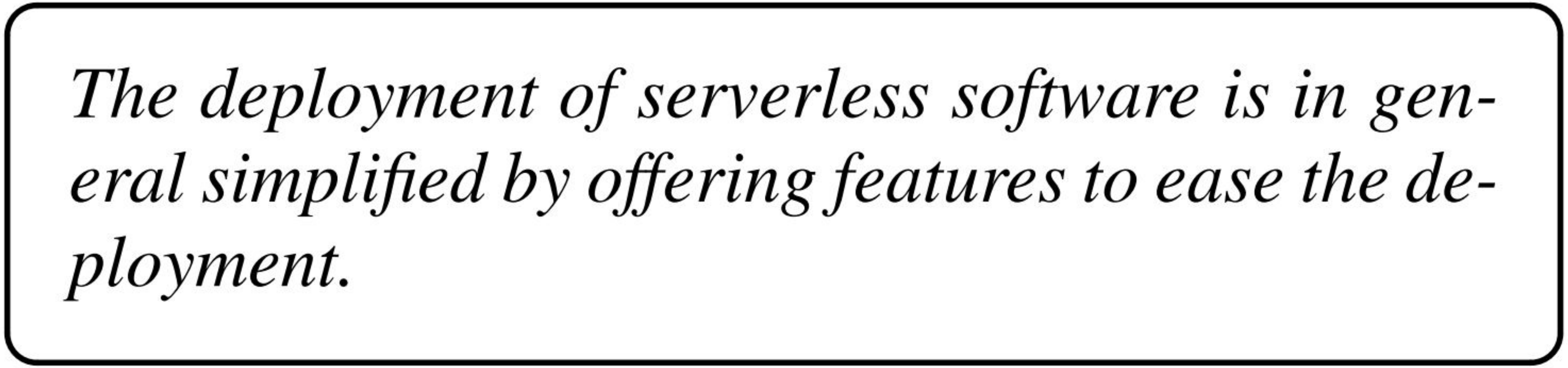
Evaluating our hypotheses

H6: The complexity of deployment of serverless software is harmful for its maintainability.

Evaluating our hypotheses

FALSE

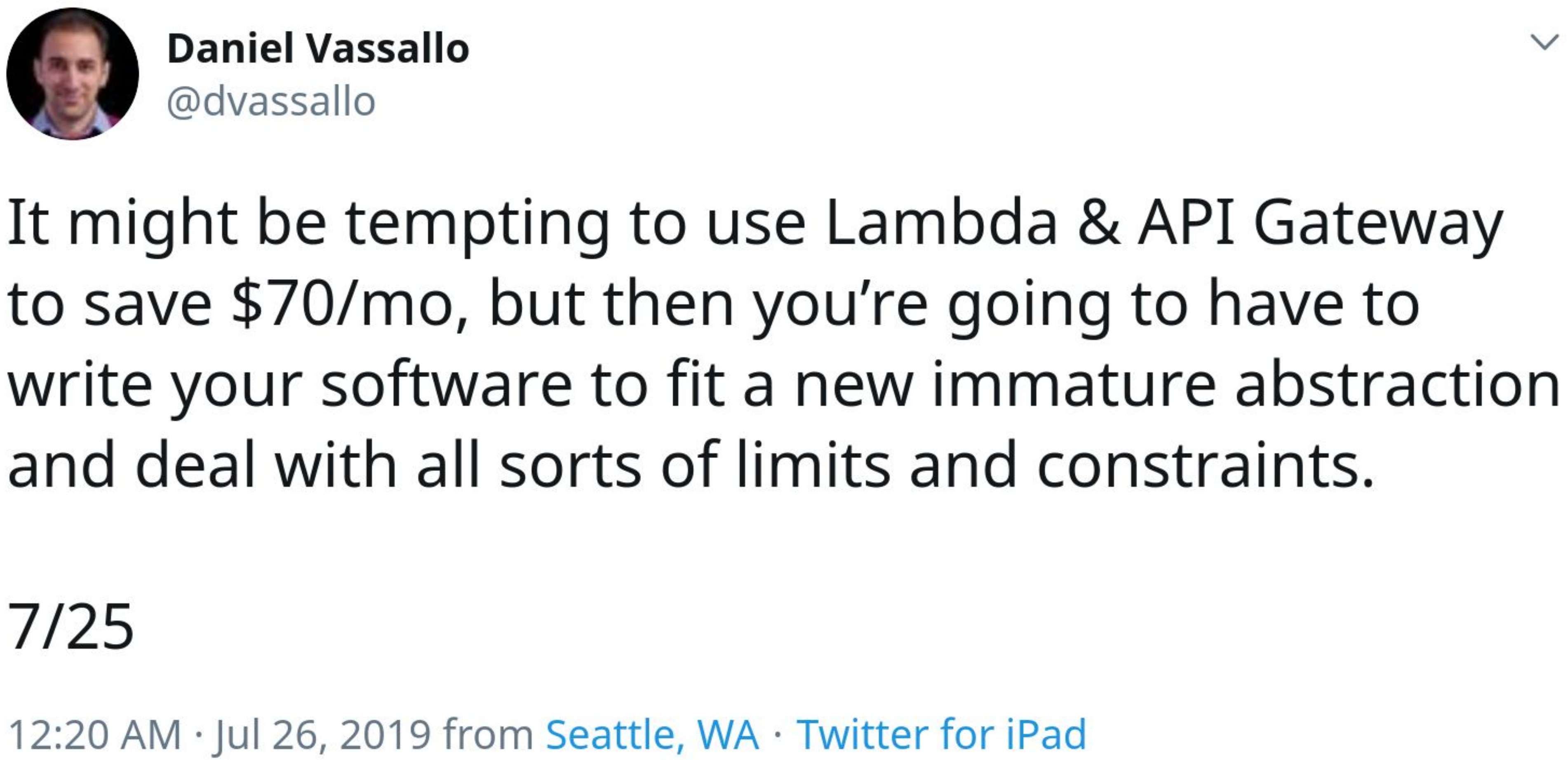
H6: The complexity of deployment of serverless software is harmful for its maintainability.



The deployment of serverless software is in general simplified by offering features to ease the deployment.

Recommendations for Serverless/FaaS projects

- FaaS is a good option for **CPU intensive tasks and event-driven applications**
- Pay special attention to **functions size and code documentation**
- Control **code duplication**
- Invest effort to **automated testing**
- Be aware of **vendor lock-in** issues



Daniel Vassallo
@dvassallo

It might be tempting to use Lambda & API Gateway to save $70/mo, but then you're going to have to write your software to fit a new immature abstraction and deal with all sorts of limits and constraints.

7/25

12:20 AM · Jul 26, 2019 from Seattle, WA · Twitter for iPad

Conclusion

- Our findings provide the current state of the projects’ maintainability using serverless architecture
- Serverless projects have a **low module coupling, their components are well balanced and are independent**
- Serverless projects tend to have **big units of code**
- **Analysability** may be an issue for serverless applications
- Serverless projects are **easy to test** because they have a small code base and they are made of independent components
- Deployment of serverless projects is in general **simplified** by platform providers
- Serverless configuration management is an **open question**
- Pay special attention to **functions size and documentation**

Research opportunities/future work

- Serverless **performance** evaluation in general and between different providers
- **cost/benefit** analysis (CBA)
- Serverless **reliability** evaluation
- Benefits and disadvantages of **configuration management** for serverless providers
- **Continuous integration** evaluation

ICSOFT 2019

14th International Conference on Software Technologies

26 - 28 July, 2019 · Prague, Czech Republic

Quality aspects of serverless architecture: an exploratory study on maintainability

Louis Racicot[1], Nicolas Cloutier[1], Julien Abt[2], Fabio Petrillo[2]

[1]*École Polytechnique de Montréal, Montréal, Canada*

[2]*Université du Québec à Chicoutimi, Chicoutimi, Canada*

{louis.racicot,nicolas.cloutier}@polymtl.com, julien.abt1@uqac.ca,fabio@petrillo.com

UQAC

Département d'informatique et mathématique

Analyzability [ISO 25010, §4.2.7.3]

“The degree of effectiveness and efficiency with which it is possible to ***assess the impact*** *on a product or system of an* ***intended change to one or more of its parts****, or to* ***diagnose*** *a product for* ***deficiencies*** *or causes of failures, or to identify parts to be modified”*

Modifiability [ISO 25010, §4.2.7.4]

“The degree to which a product or system can be effectively and efficiently modified ***without introducing defects*** *or degrading existing product quality”*

Testability [ISO 25010, §4.2.7.5]

“The degree of effectiveness and efficiency with which **test criteria can be established** for a system, product or component and tests can be **performed** to determine whether those criteria have been met.”

Modularity [ISO 25010, §4.2.7.1]

“The degree to which a system or computer program is composed of **discrete components** such that a **change to one component has minimal impact** on other components.”

Reusability [ISO 25010, §4.2.7.2]

“The degree to which an **asset** can be used in **more than one system**, or in **building other assets**.”